O'REILLY

2-е издание

Bash
Карманный справочник

ДЛЯ СИСТЕМНОГО АДМИНИСТРАТОРА



Арнольд Роббинс

2-Е ИЗДАНИЕ

Bash

Карманный справочник

SECOND EDITION

Bash

Pocket Reference

Arnold Robbins

Beijing · Boston · Farnham · Sebastopol · Tokyo O'REILLY®

2-Е ИЗДАНИЕ

Bash

Карманный справочник

Арнольд Роббинс

ДИАЛЕКТИКА

Москва · Санкт-Петербург · Киев

2017

ББК 32.973.26-018.2.75
Р58
УДК 681.3.07

Компьютерное издательство "Диалектика"

Зав. редакцией *С.Н. Тригуб*

Перевод с английского и редакция *И.В. Берштейна*

По общим вопросам обращайтесь в издательство "Диалектика" по адресу:
info@dialektika.com, http://www.dialektika.com

Роббинс, Арнольд.

Р58 Bash. Карманный справочник системного администратора, 2-е изд. : Пер. с англ. — СпБ. : ООО "Альфа-книга", 2017. — 152 с. : ил. — Парал. тит. англ.

ISBN 978-5-9909445-4-1 (рус.)

ББК 32.973.26-018.2.75

Научно-популярное издание

Арнольд Роббинс

Bash

Карманный справочник системного администратора

2-е издание

Литературный редактор	*И.А. Попова*
Верстка	*О.В. Мишутина*
Художественный редактор	*В.Г. Павлютин*
Корректор	*Л.А. Гордиенко*

Подписано в печать 24.07.2017. Формат 84x108/32.
Гарнитура Times. Усл. печ. л. 4,8. Уч.-изд. л. 5,6.
Тираж 500 экз. Заказ № 5123.

Отпечатано в АО «Первая Образцовая типография»
Филиал «Чеховский Печатный Двор»
142300, Московская область, г. Чехов, ул. Полиграфистов, д.1

ООО "И. Д. Вильямс", 127055, г. Москва, ул. Лесная, д. 43, стр. 1

ISBN 978-5-9909445-4-1 (рус.)
ISBN 978-1-491-94159-1 (англ.)

Содержание

Об авторе

Арнольд Роббинс — профессиональный программист и автор технической литературы. Он работает с системами Unix с 1980 года, а с языком Awk — с 1987 года. Будучи членом группы, принимавшей стандарт POSIX 1003.2 путем прямого голосования, он способствовал формированию этого стандарта для Awk. Арнольд специально занимается сопровождением версии GNU (`gawk`) языка Awk и документацией на него. Кроме того, он является автором четвертого издания книги *Effective awk Programming* и одним из авторов второго издания книги *Classic Shell Scripting* (обе вышли в издательстве O'Reilly).

Благодарности

Благодарю Чета Рэйми (Chet Ramey), сопровождающего Bash, за возможность получить доступ к прежним выпускам версии Bash 4.4, ответы на возникавшие вопросы и рецензирование рукописи этого справочника. Благодарю также Роберта П. Дж. Дэя (Robert P.J. Day) за рецензирование рукописи настоящего издания. Выражаю признательность Эли Зарецкому (Eli Zaretskii) за комментарии к предыдущему изданию и рецензирование рукописи настоящего издания. Наконец, благодарю Энди Орама (Andy Oram) из издательства O'Reilly за его поддержку в обновлении настоящего издания.

Благодарности из первого издания

Благодарю Чета Рэйми, сопровождающего Bash, за возможность получить доступ к прежним выпускам версии Bash 4.1 и рецензирование рукописи этого справочника. Благодарю также Роберта П. Дж. Дэя за рецензирование рукописи настоящего издания и Энди Орама из издательства O'Reilly за постоянную поддержку данного проекта.

От издательства

Вы, читатель этой книги, и есть главный ее критик и комментатор. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересно услышать и любые другие замечания, которые вам хотелось бы высказать в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо, либо просто посетить наш веб-сайт и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится или нет вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Посылая письмо или сообщение, не забудьте указать название книги и ее авторов, а также ваш обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию последующих книг.

Наши электронные адреса:

E-mail: `info@dialektika.com`
WWW: `http://www.dialektika.com`

Наши почтовые адреса:

в России: 195027, Санкт-Петербург, Магнитогорская ул., д. 30, ящик 116
в Украине: 03150, Киев, а/я 152

Карманный справочник по оболочке Bash

В этом карманном справочнике вкратце описывается оболочка Bash и, в частности, ее версия 4.4 для GNU/Linux и Mac OS X. Оболочка Bash имеется также для Solaris и различных систем BSD. Ее можно скомпилировать практически для любой другой системы Unix и даже для OpenVMS! В этом справочнике рассматриваются следующие вопросы.

- История развития оболочки.
- Краткий обзор функциональных средств.
- Вызов оболочки.
- Код завершения команды.
- Синтаксис.
- Функции.
- Переменные.
- Арифметические выражения.
- Предыстория выполнения команд.
- Автозавершение вводимых команд.
- Управление заданиями.
- Параметры оболочки.
- Выполнение команд.
- Сопроцессы.
- Ограниченные оболочки.
- Встроенные команды.
- Дополнительные источники информации.

Условные обозначения

Имена файлов, команд, параметров и приводимые примеры выделены в тексте данного справочника `моноширинным` шрифтом. Все, что должно вводиться пользователем в командной строке, выделено **`полужирным моноширинным`** шрифтом, а текст, который должен быть заменен настоящими данными в примерах и описаниях синтаксиса, — *`наклонным моноширинным`* шрифтом. Новые термины, особо подчеркиваемые слова и выражения выделяются *курсивом*. И, наконец, ссылки в форме *имя*(*N*) обозначают номер оперативной страницы руководства, где *имя* — наименование оболочки, *N* — раздел руководства, оперативно доступного по команде `man`. Имена переменных оболочки, в том числе и переменных окружения, обозначаются как `$VAR`, где `VAR` — имя переменной.

История развития оболочки

Первоначально оболочка Bourne shell (т.е. оболочка Борна) получила распространение в 1979 году вместе с системой V7 Unix и впоследствии стала стандартной для написания сценариев оболочки. Оболочку Bourne shell можно по-прежнему обнаружить в каталоге `/bin/sh` многих коммерческих систем Unix. Она не претерпела особых изменений с момента своего выпуска, хотя с годами подвергалась незначительным улучшениям. К числу наиболее примечательных новых средств, внедренных в этой оболочке, относятся переменная `CDPATH` и встроенная команда `test` (появились в Unix System III в 1980 году), хеширование команд и функции оболочки для версии System V Release 2 в 1984 году, а также средства управления заданиями, внедренные в версии System V Release 4 в 1989 году.

В оболочке Berkeley C shell (`csh`) предоставлялись средства, которые оказались более удобными для применения в диалоговом режиме (например, предыстория выполнения команд и управление заданиями), и поэтому долгое время стандартной

для программирования в Unix считалась оболочка Bourne shell, а для повседневной работы — оболочка C shell. Дэвид Корн из компании Bell Labs стал первым разработчиком, усовершенствовавшим оболочку Bourne shell, дополнив ее такими csh-подобными средствами, как предыстория команд, управление заданиями и дополнительные удобства программирования. В конечном итоге оболочка Korn shell (т.е. оболочка Корна) превзошла своим набором функциональных средств оболочки Bourne shell и C shell, сохранив в то же время совместимость с предыдущими средствами программирования на языке оболочки. А ныне язык и режим работы "стандартной оболочки" определены в стандарте POSIX на основании оболочки Bourne shell для системы Unix System V вместе с избранным поднабором функциональных средств из оболочки Korn shell.

Преследуя цель произвести систему, полностью отвечающую Unix по своим рабочим характеристикам, Фонд свободного программного обеспечения (Free Software Foundation) разработал копию оболочки Bourne shell, написанную заново и получившую название *Bash* (Bourne-Again Shell — "возрожденная" оболочка Борна). Со временем Bash стала совместимой со стандартом POSIX версией оболочки со многими дополнительными средствами, перекрывающими аналогичные средства оболочки Korn shell, хотя Bash не является точной копией оболочки Korn shell. В настоящее время оболочка Bash считается едва ли не самой широко распространенной версией, производной от оболочки Bourne shell.

Краткий обзор функциональных средств

В оболочке Bash предоставляются следующие функциональные средства и возможности.

- Переадресация ввода-вывода.
- Применение метасимволов подстановки для сокращения имен файлов.

- Переменные и параметры для специальной настройки рабочей среды.
- Встроенный набор команд для написания программ оболочки.
- Функции и оболочки для модульной организации задач, выполняемых в программах оболочки.
- Управление заданиями.
- Редактирование с использованием синтаксиса команд редактора `vi` или Emacs, имитирующее режим командной строки.
- Доступ к предыдущим командам (из предыстории их выполнения) и возможность редактировать эти команды.
- Целочисленные арифметические операции.
- Массивы и арифметические выражения.
- Применение псевдонимов для сокращения имен команд.
- Прямая совместимость со стандартом POSIX.
- Средства интернационализации.
- Цикл `for` для повторного выполнения арифметических операций.

Вызов оболочки

Интерпретатор команд оболочки Bash (`bash`) можно вызвать следующим образом:

```
bash [параметры] [аргументы]
```

Команды оболочки Bash можно выполнять с терминала, из файла (когда в качестве первого *аргумента* указан сценарий) или из стандартного ввода (если аргументы отсутствуют или указан параметр **`-s`**). Оболочка автоматически выводит приглашение на ввод команд, если стандартный ввод производится с терминала или в командной строке указан параметр **`-i`**.

Во многих системах путь к оболочке Bash осуществляется по ссылке /bin/sh. Если же оболочка Bash вызывается как sh, она действует в большей степени как традиционная оболочка Bourne shell. В частности, исходные оболочки читают содержимое файлов /etc/profile и ~/.profile, а обычные оболочки — содержимое переменной окружения $ENV, если она установлена. Подробнее об этом можно узнать на оперативной странице руководства *bash*(1).

Параметры командной строки

Во встроенной команде set (подробное ее описание приведено ниже, в разделе "Встроенные команды") могут быть указаны однобуквенные параметры командной строки, приведенные ниже.

-c *строка*. Читать команды из указанной *строки*.

-D, **-dump-strings**. Вывести из программы все символьные строки, представленные в форме $"...".

-i. Создать интерактивную оболочку (с приглашением на ввод команд). Этот параметр может и не применяться в команде set.

-l, **--login**. Обозначает такой же режим, как и для исходной оболочки.

-O *параметр*. Активизирует заданный *параметр* команды shopt. Для установки заданного *параметра* в исходное состояние служит параметр **+O**.

-p. Начать работу в качестве привилегированного пользователя. Не читать содержимое переменной окружения $ENV или $BASH_ENV; не пытаться импортировать функции из рабочей среды, а также игнорировать значения переменных BASHOPTS, CDPATH, GLOBIGNORE и SHELLOPTS. В этом случае читается содержимое обычных файлов запуска (например, ~/.bash_profile).

-r, **--restricted**. Создать ограниченную оболочку (см. далее раздел "Ограниченные оболочки").

-s. Читать команды из стандартного ввода. Результаты, выводимые из встроенных команд, направляются в файл с дескриптором **1**, а все остальные результаты, выводимые из оболочки, — в файл с дескриптором **2**.

-v, **--verbose**. Выводить строки по мере их чтения оболочкой.

--debugger. Если при запуске доступен отладочный профиль, прочитать его и активизировать параметр `extdebug` команды `shopt`. Служит для применения в отладчике оболочки Bash (подробнее об этом см. по адресу `http://bashdb.sourceforge.net`).

--dump-po-strings. То же, что и параметр **-D**, но на этот раз строки выводятся в формате `gettext` по общей лицензии GNU.

--help. Вывести сообщение об использовании команды и успешно завершить.

--init-file *файл*, **--rcfile** *файл*. Использовать указанный *файл* вместо файла `~/.bashrc` для запуска интерактивных оболочек.

--noediting. Не пользоваться библиотекой GNU Readline для ввода, даже в интерактивной оболочке.

--noprofile. Не читать файл `/etc/profile` или любые другие личные файлы запуска.

--norc. Не читать файл `~/.bashrc`. Активизируется автоматически, когда оболочка вызывается как `sh`.

--posix. Активизировать режим работы по стандарту POSIX.

--version. Вывести сообщение об используемой версии оболочки и успешно завершить.

-, **--**. Завершить обработку параметров.

Пояснение остальных параметров приведено далее, в разделе "Встроенные команды" при описании команды `set`.

Аргументы

Аргументы присваиваются позиционным параметрам `$1`, `$2` и т.д. Если в качестве первого аргумента указывается сценарий, из него читаются команды, а остальные аргументы присваиваются позиционным параметрам `$1`, `$2` и т.д. Имя сценария доступно в качестве позиционного параметра `$0`. Файл самого сценария не обязательно должен быть исполняемым, но доступным для чтения.

Код завершения команды

По завершении любой команды предоставляется числовой *код* ее *завершения* или *возвращаемое значение*. Внешние команды вроде `ls` предоставляют такое значение операционной системе. А внутренние команды вроде `cd` предоставляют его непосредственно оболочке.

Оболочка автоматически извлекает возвращаемое значение по завершении команды. По общепринятому соглашению нулевой код завершения означает *истинный* или *удачный* исход выполнения команды, а любой другой код завершения — *ложный* или *неудачный* исход ее выполнения. Именно таким образом в оболочке организуется управление потоком команд в таких операторах, как `if`, `while` и `until`.

Кроме того, оболочка делает значение, возвращаемое последней выполнявшейся командой, доступным для своего сценария в переменной `$?`. Но, как правило, это значение следует сохранять в другой переменной, поскольку последующие команды из сценария перезапишут его в переменной `$?`.

Значения кодов завершения могут находиться в пределах от **0** до **255**. Для обозначения отдельных условий завершения команд в оболочке применяются конкретные значения, перечисленные ниже.

Числовое значение кода завершения	Назначение
`0`	Обозначает удачный исход выполнения команды
`2`	Возвращается встроенными командами для указания на ошибки в их использовании
`126`	Обозначает, что команда была обнаружена, но не выполнена
`127`	Обозначает, что команда не была обнаружена
`128+N`	Обозначает, что выполнение команды было прервано, поскольку был получен сигнал ***N***

Синтаксис

В этом разделе описываются многие обозначения, присущие рассматриваемой здесь оболочке. В нем поочередно рассматриваются следующие вопросы:

- Специальные файлы.
- Метасимволы подстановки имен файлов.
- Раскрытие скобок.
- Управляющие последовательности символов.
- Заключение в кавычки.
- Формы команд.
- Формы переадресации ввода-вывода.

Специальные файлы

Оболочка читает содержимое одного или нескольких файлов запуска. Содержимое некоторых из этих файлов читается только в том случае, если оболочка является исходной, т.е. служит для входа в систему. Оболочка Bash читает содержимое этих файлов в следующем порядке.

- Файл `/etc/profile`. Выполняется автоматически при входе в систему.

- Первый файл, обнаруживаемый в следующем списке: `~/.bash_profile`, `~/.bash_login` или `~/.profile`. Выполняется автоматически при входе в систему.
- Файл `~/.bashrc`. Читается всякой неисходной оболочкой. Но если оболочка Bash вызывается как `sh` или с параметром **`--posix`**, то она читает содержимое переменной окружения `$ENV` ради обеспечения совместимости со стандартом POSIX.

Функции `getpwnam()` и `getpwuid()` из библиотеки C служат для выдачи информации о начальных каталогах по сокращениям ~*имя*. (В персональных системах база данных пользователя хранится в файле `/etc/passwd`. Но в сетевых системах эта информация может поступать из служб NIS, NIS+, LDAP и других источников, а не из файла паролей на рабочей станции пользователя.)

Когда происходит выход из интерактивной исходной оболочки или же в неинтерактивной исходной оболочке выполняется встроенная команда `exit`, оболочка Bash читает и выполняет содержимое файла `~/.bash_logout`, при условии, что этот файл существует. (Исходной является та оболочка, где установлен параметр **`-l`**.)

Метасимволы подстановки имен файлов

Ниже перечислены метасимволы, применяемые для подстановки имен файлов.

`*`	Совпадение с любой строкой, содержащей от нуля и больше символов
`?`	Совпадение с любым символом
`[`***abc***`...]`	Совпадение с любым символом в квадратных скобках, где дефис может обозначать заданный диапазон символов (например, **a-z**, **A-Z**, **0-9**)
`[!`***abc***`...]`	Совпадение с любым символом, кроме тех, которые указаны в квадратных скобках
`~`	Начальный каталог текущего пользователя
~*имя*	Начальный каталог пользователя, имеющего указанное ***имя***
`~+`	Текущий рабочий каталог (содержимое переменной **$PWD**)
`~-`	Предыдущий рабочий каталог (содержимое переменной **$OLDPWD**)

Если установлен параметр `extglob`, то для подстановки имен файлов применяются следующие метасимволы.

`?(`***шаблон***`)`	Совпадение с нулевым или единичным количеством экземпляров заданного ***шаблона***
`*(`***шаблон***`)`	Совпадение с нулевым или большим количеством экземпляров заданного ***шаблона***
`+(`***шаблон***`)`	Совпадение с единичным или большим количеством экземпляров заданного ***шаблона***
`@(`***шаблон***`)`	Точное совпадение с одним экземпляром заданного ***шаблона***
`!(`***шаблон***`)`	Совпадение с любыми символьными строками, *не* совпадающими с заданным ***шаблоном***

В качестве заданного *шаблона* может служить последовательность шаблонов, разделяемая знаком `|`, который обозначает, что совпадение может произойти с любым из перечисленных шаблонов. Этот расширенный синтаксис напоминает синтаксис сопоставления с шаблонами, доступный в командах `egrep` и `awk`.

Если активизирован параметр `globstar`, то для подстановки имен файлов применяется следующий метасимвол.

`**`	Совпадение со всеми файлами или нулевым или большим количеством подкаталогов. Если дополнительно указан знак косой черты, то совпадение происходит только с каталогами и подкаталогами

В оболочке Bash поддерживается обозначение `[[=`*c*`=]]`, принятое в стандарте POSIX для указания на совпадение с символами, которые имеют одинаковый вес, а также обозначение `[[.`*c*`.]]` для последовательностей сортировки. Кроме того, форма `[[:`*класс*`:]]` позволяет обозначить совпадение с перечисленными ниже классами символов.

Класс	Совпадающие символы
alnum	Буквенно-цифровые
alpha	Буквенные
ascii	Символы в коде ASCII (не по стандарту POSIX)
blank	Пробелы и знаки табуляции
cntrl	Управляющие
digit	Десятичные цифры
graph	Непробельные

Класс	Совпадающие символы
lower	Строчные буквы
print	Печатаемые
punct	Знаки препинания
space	Пробельные
upper	Прописные буквы
word	[[:word:]] означает то же самое, что и [[:alnum:]] (не по стандарту POSIX)
xdigit	Шестнадцатеричные цифры

Совет

Сценарии читаются в оболочке Bash построчно. Каждая прочитанная строка полностью подвергается синтаксическому анализу, прежде чем начать выполнение любой из команд в этой строке. Такой подход чреват следующими последствиями.

- Нельзя определить псевдоним и воспользоваться им в той же самой строке.
- Команды, оказывающие непосредственное влияние на синтаксический анализ сценария, должны быть указаны в отдельных строках прежде тех частей сценария, на анализ которых они могут оказать влияние.

То же самое касается и функций. Они подвергаются синтаксическому анализу все сразу, и поэтому параметр `extglob` нельзя активизировать в теле функции, ожидая, что это повлияет только на саму функцию. Следовательно, чтобы воспользоваться расширенными возможностями сопоставления с шаблонами, следует ввести команду с этим параметром в отдельной строке, расположив ее в самом начале сценария, как показано ниже.

```
shopt -s extglob     # активизировать расширенные
                     #  шаблоны оболочки
```

Примеры

```
ls new*              # Перечислить файлы new и new.1
cat ch?              # Обнаружить совпадение
                     # с последовательностью символов ch9,
                     # но не ch10
```

```
gvim [D-R]*                  # Обнаружить совпадение
                             # с именами файлов,
                             # начинающимися на буквы от D до R
pr !(*.o|core) | lpr         # Вывести файлы, не являющиеся
                             # объектными и не относящиеся
                             # к ядру системы
```

ПРИМЕЧАНИЕ

В современных системах диапазоны символов вроде **`[D-R]`** не являются переносимыми. Региональные настройки системы могут включать в себя не только прописные буквы от **`D`** до **`R`** в указанном диапазоне символов. Впрочем, для контроля над этой ситуацией можно воспользоваться параметром **`globasciiranges`** (см. его описание далее, в разделе "Параметры оболочки").

Раскрытие скобок

Поддержка раскрытия скобок в оболочке Bash уже давно основывается на аналогичном свойстве оболочки C shell. В отличие от символов подстановки имен файлов, раскрытие скобок имеет исключительно текстовый характер. В частности, слова, образующиеся при раскрытии скобок, совсем не обязательно должны совпадать с именами существующих файлов. Имеются следующие формы раскрытия скобок.

pre*`{`*X*`,`*Y*`[,`*Z*...`]}`*post. Раскрывается до `preXpost`, `preYpost` и т.д.

pre*`{`*start*`..`*end*`[..`*incr*`]}`*post, где ***start*** и ***end*** обозначают целые числа или одиночные буквы, а ***incr*** — целое число. Оболочка расширяет эту конструкцию до полного диапазона в пределах от ***start*** до ***end***, увеличивая целое число ***incr***, если оно указывается.

Постфиксный и префиксный тексты в обеих формах необязательны. В числовом выражении пределы ***start*** и ***end*** вместе или по отдельности могут предваряться одним или более начальным нулем. А результаты раскрытия скобок дополняются нулями до максимальной ширины пределов ***start*** и ***end***. Оболочка Bash игнорирует начальные нули в целом числе ***incr***, всегда интерпретируя его как десятичное значение.

Раскрытие скобок может быть вложенным, а его результаты — *не* отсортированными. Раскрытие скобок выполняется прежде других видов раскрытия выражений, а открывающие и закрывающие фигурные скобки не должны заключаться в кавычки, чтобы оболочка Bash смогла распознать их. Подстановки команд при раскрытии скобок игнорируются оболочкой Bash. Чтобы избежать конфликта с подстановкой параметров, раскрытие скобок не должно начинаться с последовательности символов **${**.

Примеры

```
# раскрыть текстуально, без сортировки
$ echo hi{DDD,BBB,CCC,AAA}there
hiDDDthere hiBBBthere hiCCCthere hiAAAthere

# раскрыть и найти совпадение с именами
# файлов ch1, ch2, app1, app2
$ ls {ch,app}?

# раскрыть до mv info info.old
$ mv info{,.old}

# простая числовая подстановка
$ echo 1 to 10 is {1..10}
1 to 10 is 1 2 3 4 5 6 7 8 9 10

# числовая подстановка с приращением
$ echo 1 to 10 by 2 is {1..10..2}
1 to 10 by 2 is 1 3 5 7 9

# числовая подстановка с дополнением нулями
$ echo 1 to 10 with zeros is {01..10}
1 to 10 with zeros is 01 02 03 04 05 06 07 08 09 10
```

Управляющие последовательности символов

В оболочке Bash *управляющие последовательности символов* распознаются и интерпретируются в следующих контекстах.

- Символьная строка в форме `$'...'`, заключаемая в одиночные кавычки.
- Аргументы, указываемые в командах `echo -e` и `printf %b`.
- Форматирующие строки, указываемые в команде `printf`.

В приведенной ниже таблице перечислены наиболее употребительные управляющие последовательности символов, принятые во всех контекстах, а также управляющие последовательности символов, особенные для каждого из упомянутых выше контекстов.

Управляющая последовательность	Доступность	Назначение
`\a`	Все контексты	Звонок (BEL в коде ASCII) — звуковой или визуальный предупреждающий сигнал
`\b`	Все контексты	Возврат на одну позицию
`\c`	Команды `echo -e` и `printf %b`	Подавить завершающий знак новой строки (например, в команде `echo -n`) и не выводить ни один из последующих символов
`\c`*X*	Строка в форме `$'...'`	Управляющий символ *X*
`\e`	Все контексты	Переход
`\E`	Все контексты	Переход
`\f`	Все контексты	Перевод страницы
`\n`	Все контексты	Перевод строки
`\r`	Все контексты	Возврат каретки
`\t`	Все контексты	Табуляция
`\u`*HHHH*	Все контексты	Символ *HHHH* в Юникоде
`\u`*HHHHHHHH*	Все контексты	Символ *HHHHHHHH* в Юникоде
`\v`	Все контексты	Вертикальная табуляция

Управляющая последовательность	Доступность	Назначение
`\xHH`	Все контексты	Шестнадцатеричное значение *HH*
`\nnn`	Строка в форме `$'...'`, команда `printf`	Восьмеричное значение *nnn*
`\0nnn`	Команды `echo -e` и `printf %b`	Восьмеричное значение *nnn*
`\'`	Строка в форме `$'...'`	Одиночная кавычка
`\"`	Строка в форме `$'...'`	Двойная кавычка
`\?`	Строка в форме `$'...'`	Знак вопроса
`\\`	Все контексты	Обратная косая черта

Кроме того, оболочка интерпретирует несколько перекрывающийся набор управляющих последовательностей символов, преобразуя их в соответствующие значения переменных `PS0`, `PS1`, `PS2` и `PS4` для строк приглашения. Более подробно этот вопрос обсуждается далее, в разделе "Специальные строки приглашений".

Заключение в кавычки

Благодаря заключению в кавычки отменяется специальное назначение символов и появляется возможность использовать их буквально. В приведенной ниже таблице перечислены символы, имеющие специальное назначение.

Символ	Назначение
`;`	Разделитель команд
`&`	Выполнение команд в фоновом режиме
`()`	Группирование команд
`|`	Канал
`< > &`	Знаки переадресации
`* ? [ ] ~ + - @ !`	Метасимволы подстановки имен файлов
`" ' \`	Служат для заключения в кавычки других символов
`` ` ``	Подстановка команд

Символ	Назначение
$	Подстановка переменных (команд или арифметических выражений)
#	Обозначает начало комментария, продолжающегося до конца строки
Знаки пробела, табуляции и новой строки	Разделители строк

В кавычки могут быть заключены следующие символы.

"..." Все, что заключено в открывающие (") и закрывающие (") кавычки, за исключением перечисленных ниже символов, сохраняющих свое специальное назначение

- **$** Подстановка (переменных, команд или арифметическая подстановка).
- ` Подстановка команд.
- " Обозначает конец символьной строки, заключаемой в двойные кавычки

'...' Все, что заключено в открывающие (') и закрывающие (') одиночные кавычки, воспринимается буквально, за исключением другого знака одиночной кавычки (')

\ Символ, следующий после знака \, воспринимается буквально. Для экранирования знаков ", **$** и ` их следует заключать в двойные кавычки: "..."

$"..." То же, что и "...", за исключением преобразования по языковому стандарту

$'...' То же, что и '...', только текст в одиночных кавычках обрабатывается с учетом управляющих последовательностей символов, как пояснялось ранее в разделе "Управляющие последовательности символов"

Примеры

```
$ echo 'Single quotes "protect" double quotes'
Single quotes "protect" double quotes
$ echo "Well, isn't that \"special\"?"
Well, isn't that "special"?
$ echo "You have `ls | wc -l` files in `pwd`"
You have 43 files in /home/bob
$ echo "The value of \$x is $x"
The value of $x is 100
$ echo $'A\tB'
A       B
```

Формы команд

Ниже приведены различные формы команд, допустимые в оболочке Bash.

`cmd &`	Выполнить указанную команду ***cmd*** в фоновом режиме
`{ cmd1 ; cmd2 ; }`	Выполнить указанные команды группой в текущей оболочке
`( cmd1 ; cmd2 ; )`	Выполнить указанные команды группой в подоболочке
`cmd1 | cmd2`	Передать выход из команды ***cmd1*** по каналу на вход команды ***cmd2***
``cmd1 `cmd2` ``	Произвести подстановку команд. В частности, использовать результат, выводимый из команды ***cmd2***, в качестве аргумента команды ***cmd1***
`cmd1 $(cmd2)`	Произвести подстановку команд по стандарту POSIX. Допускается вложение
`cmd $((выражение))`	Произвести арифметическую подстановку по стандарту POSIX. В частности, использовать числовой результат вычисления заданного ***выражения*** в качестве аргумента указанной команды ***cmd***
`cmd1 && cmd2`	Выполнить логическую операцию И. В частности, выполнить сначала указанную команду ***cmd1***, а затем команду ***cmd2*** при удачном исходе выполнения команды ***cmd1***. Это "укороченная" форма логической операции, при которой команда ***cmd2*** вообще не выполняется при *неудачном* исходе выполнения команды ***cmd1***
`cmd1 || cmd2`	Выполнить логическую операцию ИЛИ. В частности, выполнить указанную команду ***cmd1***, а при неудачном исходе ее выполнения — команду ***cmd2***. Это "укороченная" форма логической операции, при которой команда ***cmd2*** вообще не выполняется при *удачном* исходе выполнения команды ***cmd1***
`! cmd`	Выполнить логическую операцию НЕ. В частности, выполнить указанную команду ***cmd*** и выдать нулевой код завершения, если выполнение команды ***cmd*** завершится ненулевым кодом. В противном случае выдать ненулевой код завершения, если выполнение команды ***cmd*** завершится нулевым кодом

Примеры

```
# Выполнить указанные команды в фоновом режиме
$ nroff file > file.txt &

# Выполнить указанные команды последовательно
$ cd; ls

# Переадресовать все выводимые результаты
$ (date; who; pwd) > logfile

# Отсортировать файл, подготовить, а затем произвести
# постраничный вывод полученных результатов
$ sort file | pr -3 | lpr

# Отредактировать файлы, обнаруженные командой grep
$ gvim `grep -l ifdef *.cpp`

# Указать список искомых файлов
$ egrep '(yes|no)' `cat list`

# Версия предыдущей формы команды по стандарту POSIX
$ egrep '(yes|no)' $(cat list)

# Сделать то же самое, но быстрее, хотя и не по-стандарту
# POSIX
$ egrep '(yes|no)' $(< list)

# Вывести файл, если он содержит заданный шаблон.
# Сделать это негласно, направив выводимый результат и
# сообщения об ошибках в файл /dev/null
$ grep XX file > /dev/null 2>&1 && lpr file

# С другой стороны, отобразить эхом сообщение об ошибке
$ grep XX file || echo "XX not found"
```

Формы переадресации ввода-вывода

Ниже приведены различные формы переадресации ввода-вывода, допустимые в оболочке Bash.

Дескриптор файла	Наименование	Общепринятое сокращение	Стандартное устройство ввода-вывода
0	Стандартный ввод	`stdin`	Клавиатура
1	Стандартный вывод	`stdout`	Экран терминала
2	Стандартный вывод ошибок	`stderr`	Экран терминала

Обычный источник ввода или адресат вывода можно изменить, как поясняется в последующих подразделах.

Простая переадресация ввода-вывода

Ниже приведены формы простой переадресации ввода-вывода.

cmd > ***файл***. Направить результат, выводимый из указанной команды *cmd*, в заданный ***файл***, фактически перезаписав его содержимое.

cmd >> ***файл***. Направить результат, выводимый из указанной команды *cmd*, в заданный ***файл***, присоединив его к содержимому данного файла.

cmd < ***файл***. Взять входные данные для указанной команды *cmd* из заданного ***файла***.

cmd < ***текст***. Все содержимое сценария оболочки вплоть до строки, соответствующей заданному ***тексту***, становится стандартным вводом для указанной команды *cmd*, причем заданный ***текст*** может храниться в переменной оболочки. Такую форму команд иногда еще называют *встраиваемым документом*, где входные данные вводятся с клавиатуры или из программы оболочки. Подобный синтаксис обычно применяется в командах `cat`, `ex` и `sed`. (Если для переадресации ввода используется форма **<<-**, начальные знаки табуляции удаляются из содержимого встраиваемого документа, а остальные знаки табуляции игнорируются при сравнении входных данных с признаком конца ввода заданного ***текста***.) Если же какая-нибудь часть заданного ***текста*** заключается в кавычки, входные

данные передаются буквально. В противном случае обрабатывается содержимое, получаемое в результате подстановки переменных, команд и арифметических выражений.

cmd* <<< *слово. Предоставить текст заданного ***слова*** вместе с завершающим знаком новой строки в качестве входных данных для указанной команды ***cmd***. (Это так называемая *встраиваемая строка* — термин, заимствованный из свободно доступной версии оболочки `rc`, см. также раздел "Дополнительные источники информации".)

cmd* <> *файл. Открыть заданный ***файл*** для чтения *и* записать в него результат выполнения указанной команды ***cmd***, направляемый в стандартный вывод. Прежнее содержимое заданного ***файла*** *не* нарушается[1].

cmd* <| *файл. Направить результат, выводимый указанной командой ***cmd***, в заданный ***файл***, перезаписав его содержимое, даже если в оболочке установлен параметр `noclobber`.

Переадресация ввода-вывода с использованием дескрипторов файлов

Ниже приведены формы переадресации ввода-вывода с использованием дескрипторов файлов.

cmd* >&*n. Направить результат, выводимый указанной командой ***cmd***, в файл с заданным дескриптором ***n***.

cmd m*>&*n. То же, что и выше, за исключением того, что выводимый результат, обычно направляемый в файл с заданным дескриптором ***m***, фактически направляется в файл с дескриптором ***n***.

***cmd* >&-**. Закрыть стандартный вывод.

---

[1] Если переадресация ввода обозначается как **<**, то заданный файл открывается только для чтения, а попытка записи по дескриптору этого файла завершится неудачно. А если переадресация ввода обозначается как **<>**, то заданный файл открывается как для чтения, так и для записи. И этим можно выгодно воспользоваться в приложении.

cmd* <&*n. Взять входные данные для указанной команды ***cmd*** из файла с заданным дескриптором ***n***.

cmd* *m*<&*n. То же, что и выше, за исключением того, что входные данные, обычно поступающие из файла с заданным дескриптором ***m***, на этот раз поступают из файла с дескриптором ***n***.

***cmd*<&-**. Закрыть стандартный ввод.

cmd*<&*n*-**. Направить содержимое файла с заданным дескриптором ***n в стандартный ввод, получив сначала копию, а затем закрыв оригинал.

cmd*>&*n*-**. Направить содержимое файла с заданным дескриптором ***n в стандартный вывод, получив сначала копию, а затем закрыв оригинал.

Многократная переадресация ввода-вывода

Ниже приведены формы многократной переадресации ввода-вывода.

cmd* 2> *файл. Направить стандартный вывод ошибок в заданный ***файл***. При этом стандартный вывод данных остается прежним (в частности, он направляется на экран терминала).

cmd* > *файл* 2>&1**. Направить стандартный вывод данных и ошибок в заданный ***файл.

cmd*>& *файл. То же, что и выше.

cmd* &> *файл. То же, что и выше, но более предпочтительная форма.

cmd* &>> *файл. Присоединить стандартный вывод данных и ошибок к содержимому заданного ***файла***.

cmd* > *файл*$_1$ 2> *файл*$_2$**. Направить стандартный вывод данных в заданный ***файл$_1$, а стандартный вывод ошибок — в указанный ***файл***$_2$.

cmd* | tee *файлы. Направить результат, выводимый указанной командой ***cmd***, как в стандартный вывод (как

правило, на терминал), так и в заданные ***файлы***. См. оперативную страницу руководства *tee*(1).

cmd* `2>&1 | tee` *файлы. Направить стандартный вывод данных и ошибок из указанной команды ***cmd*** по каналу команде **`tee`** для переадресации как в стандартный вывод (как правило, на терминал), так и в заданные ***файлы***.

cmd* `|& tee` *файлы. То же, что и выше.

В оболочке Bash допускаются многоразрядные числовые обозначения дескрипторов файлов, не требующие специального синтаксиса. А в большинстве других оболочек для этой цели требуется специальный синтаксис или же подобная возможность вообще не допускается.

ПРИМЕЧАНИЕ

Между дескрипторами файлов и знаком переадресации ввода-вывода пробелы *не* допускаются. В остальных случаях дополнительные пробелы допускаются, хотя они и не обязательны.

Подстановка процессов

Ниже приведены формы переадресации ввода-вывода с подстановкой процессов.

cmd* `<(`*команда*`)`**. Выполнить заданную ***команду, соединив ее выход с именованным каналом или файлом, открытым по пути `/dev/fd`, а также поместить имя файла в список аргументов указанной команды ***cmd***. Указанная команда ***cmd*** может прочитать содержимое этого файла, чтобы обнаружить в нем результат, выводимый заданной ***командой***.

cmd* `>(`*команда*`)`**. Выполнить заданную ***команду, соединив ее вход с именованным каналом или файлом, открытым в каталоге `/dev/fd`, а также поместить имя файла

в список аргументов указанной команды **cmd**. Результат, выводимый указанной командой **cmd** в этот файл, служит в качестве входных данных для заданной ***команды***.

Подстановка процессов доступна в тех системах, где поддерживаются именованные каналы обратного магазинного типа (FIFO) или доступ к открытым файлам по их именам в каталоге `/dev/fd`. (Это справедливо для всех современных систем Unix.) Подобным образом предоставляется возможность создавать нелинейные каналы. Подстановка процессов недоступна в оболочках, работающих в режиме по стандарту POSIX.

Сохранение дескрипторов файлов в переменных

В оболочке Bash допускается указывать форму {*имя_переменной*} вместо числового обозначения дескриптора файла при переадресации ввода-вывода. В подобных случаях оболочка выбирает в качестве дескриптора файла числовое значение свыше **9**, присваивая его именованной переменной оболочки. Указанное *имя_переменной* может обозначать элементы массива и переменные, имеющие специальное назначение в оболочке. Например:

```
# Сохранить числовое обозначение дескриптора файла
$ echo foo {foofd}> /tmp/xyzzy
foo
$ echo $foofd
11
```

Подобная форма чаще всего применяется при переадресации ввода-вывода в команде `exec`. Благодаря этому дескриптор файла, сохраненный в переменной оболочки, может быть далее использован в сценарии.

ПРИМЕЧАНИЕ

Открыв файл по его дескриптору подобным образом, вы должны закрыть его сами. Оболочка Bash не сделает это за вас.

Специальные имена файлов

При переадресации ввода-вывода в оболочке Bash распознается несколько специальных имен файлов. Такие имена получают внутреннюю интерпретацию в оболочке Bash только в том случае, если они отсутствуют в самой системе. Ниже перечислены специальные имена файлов, употребляемые в оболочке Bash.

/dev/stdin. Дубликат дескриптора файла **0**.

/dev/stdout. Дубликат дескриптора файла **1**.

/dev/stderr. Дубликат дескриптора файла **2**.

/dev/fd/<*n*>. Дубликат дескриптора файла **<*n*>**.

dev/tcp/<*host*>/<*port*>. Оболочка Bash устанавливает соединение с указанным хостом (т.е. сетевым узлом), где **<*host*>** означает имя или IP-адрес хоста, через заданный порт **<*port*>**, используя полученный в итоге дескриптор файла при переадресации ввода-вывода.

dev/udp/<*host*>/<*port*>. Оболочка Bash устанавливает соединение с указанным хостом, где **<*host*>** означает имя или IP-адрес хоста, через заданный порт **<*port*>**, используя полученный в итоге дескриптор файла при переадресации ввода-вывода.

Примеры

```
# Копировать файл part1 в файл book
$ cat part1 > book

# Присоединить файлы part2 и part3
$ cat part2 part3 >> book

# Отправить отчет большому начальнику
$ mail tim < report

# Встраиваемый документ составляется ниже из входных данных,
# направляемых команде sed
$ sed 's/^/XX /g' << END_ARCHIVE
> Именно так архив оболочки зачастую "сворачивается",
> увязывая текст для распространения. Как правило, sed
```

```
> выполняется из программы оболочки, а не из командной
> строки.
> END_ARCHIVE
XX Именно так архив оболочки зачастую "сворачивается",
XX увязывая текст для распространения. Как правило, sed
XX выполняется из программы оболочки, а не из командной
XX строки.
```

Чтобы переадресовать стандартный вывод данных в стандартный вывод ошибок, достаточно сделать следующее:

```
$ echo "Usage error: see administrator" 1>&2
```

В следующем примере выводимый из команды результат (обнаруженные файлы) направляется в указанный файл `filelist`, а сообщения об ошибках (в связи с недоступными файлами) — в файл `no_access`:

```
$ find / -print > filelist 2> no_access
```

А в приведенном ниже примере сортируются два файла и по команде `diff` предоставляются отличия в получаемых результатах.

```
$ diff -u <(sort file1) <(sort file2) | less
```

Функции

Функция в оболочке представляет собой совокупность команд, выполняемых в сценарии оболочки. Функции способствуют модульной организации программ оболочки, разделяя их на отдельные задачи. Благодаря этому код выполнения каждой задачи не повторяется всякий раз, когда требуется выполнить ее. Синтаксис определения функции в оболочке по стандарту POSIX соответствует следующему синтаксису, принятому в оболочке Bourne shell:

```
имя () {
    здесь следует код, образующий тело функции
} [виды переадресации]
```

Функции вызываются таким же образом, как и обычные команды, встроенные в оболочку или внешние по отношению к ней. Позиционные параметры $1, $2 и т.д. получают аргументы из командной строки, временно скрывая значения глобальных переменных $1, $2 и т.д., тогда как имя всего сценария оболочки остается в переменной $0. Ниже приведен пример определения и вызова функции в оболочке.

```
# Функция fatal - выдать сообщение о неисправимой
# ошибке и прервать выполнение:

fatal () {
    # Сообщения направляются в стандартный вывод ошибок
    echo "$0: fatal error:" "$@" >&2
    exit 1
}
...
if [ $# = 0 ]      # недостаточно аргументов
then
    fatal not enough arguments
fi
```

Для возврата кода завершения вызывающей части программы оболочки в функции может использоваться команда `return`. В соответствии со стандартом POSIX любые виды переадресации ввода-вывода, задаваемые в определении функции, вычисляются при ее выполнении, а не при ее определении. В оболочке Bash допускается определять функции, используя несколько иной синтаксис, приведенный ниже. Если в определении функции используется ключевое слово `function`, то указывать скобки после имени функции необязательно.

`function` *имя* `[()] {` *тело функции* `}` `[`*виды переадресации*`]`

Функции, в именах которых отсутствует знак **=** или **/**, могут быть экспортированы в рабочую среду по команде `export -f`. Подробнее об этом см далее описание команды `export` в разделе “Встроенные команды”.

Функции разделяют общие с “родительской” оболочкой прерывания. Эти прерывания вкратце описываются ниже (см. также описание команды `trap` далее в разделе “Встроенные команды”).

Тип прерывания	Общее/Необщее
Прерывание по сигналу	Остается общим до тех пор, пока не будет переопределено в самой функции
DEBUG	Является необщим, если только не активизирован режим трассировки функций (по команде **set -T** или **set -o functrace**). Если этот режим не активизирован, прерывание **DEBUG**, формируемое при вызове функции, остается на месте после возврата из функции
ERR	Является необщим, если только не активизирован режим трассировки ошибок (по команде **set -T** или **set -E errtrace**)
EXIT	Остается общим до тех пор, пока не будет переопределено в самой функции
RETURN	Является необщим, если только не активизирован режим трассировки функций (по команде **set -T** или **set -o functrace**)

Функции могут иметь локальные переменные и могут быть рекурсивными. Но, в отличие от оболочки Korn shell, синтаксис для определения функций в оболочке Bash для этих целей не подходит.

Имена функций совсем не обязательно должны быть достоверными в оболочке идентификаторами (как, впрочем, имена внешних команд). Но это правило не распространяется на оболочки с режимом работы по стандарту POSIX. Кроме того, в подобных оболочках не разрешается определять функции с такими же именами, как и у специальных встроенных команд. Ведь это было бы опрометчивостью в интерактивных оболочках и непоправимой ошибкой в неинтерактивных оболочках.

В оболочке Bash применяется модель *соблюдения динамических областей видимости*, где переменные, объявляемые как локальные, доступны не только в теле функции, но и в других функциях, которые ее вызывают. Этим данная оболочка заметно отличается от многих других оболочек типа Bourne.

Совет

Ни в коем случае *не* пользуйтесь командой `exit` в теле функции, кроме тех случаев, когда действительно требуется прервать выполнение всей программы.

Переменные

В этом разделе рассматривающие следующие вопросы, касающиеся переменных.

- Присваивание значений переменным.
- Подстановка переменных.
- Косвенные переменные.
- Переменные, встроенные в оболочку.
- Другие переменные оболочки.
- Массивы.
- Специальные строки приглашений.

Присваивание значений переменным

Имена переменных могут состоять из любых букв, цифр и знаков подчеркивания и они не должны начинаться с цифры. Прописные и строчные буквы различаются в именах переменных. Значения присваиваются переменным с помощью операции **=**. Присваиваемое значение *не* должно отделяться пробелами от имени переменной. В одной строке можно присвоить значения сразу нескольким переменным, отделив их друг от друга пробелами:

```
firstname=Arnold lastname=Robbins numkids=4 numpets=1
```

Имена переменных, применяемых или устанавливаемых в оболочке, принято обозначать *только* прописными буквами. Хотя переменные с именами, набранными только прописными

буквами, можно применять и в сценариях, при условии, что они не совпадают с именами специальных переменных оболочки.

По умолчанию оболочка интерпретирует значения переменных как строковые, даже если они состоят только из цифр. Но если значение присваивается целочисленной переменной, создаваемой по команде `declare -i`, то оболочка Bash интерпретирует правую часть операции присваивания как выражение (подробнее об этом см. далее раздел "Арифметические выражения"). Например:

```
$ i=5+3 ; echo $i
5+3
$ declare -i jj ; jj=5+3 ; echo $jj
8
```

В операции **+=** правая ее часть прибавляется к существующему значению переменной и затем присваивается ей. В этом случае целочисленные переменные интерпретируют правую часть данной операции присваивания как выражение, которое вычисляется, а его результат прибавляется к существующему значению переменной и затем присваивается переменной. При этом в массивы вводятся новые элементы (подробнее об этом см. далее в разделе "Массивы"). Например:

```
$ name=Arnold                    # Строковая переменная
$ name+=" Robbins" ; echo $name
Arnold Robbins
$ declare -i jj ; jj=3+5         # Целочисленная переменная
$ echo $jj
8
$ jj+=2+4 ; echo $jj
14
$ pets=(blacky rusty)             # Переменная массива
$ echo ${pets[*]}
blacky rusty
$ pets+=(raincloud sophie)
$ echo ${pets[*]}
blacky rusty raincloud sophie
```

Подстановка переменных

В приведенных ниже выражениях пробелы не допускаются. Дополнительно указывать двоеточие (`:`) необязательно, а если оно указано, то *переменная* должна быть непустой и установленной. В формах подстановки переменных во внимание принимается значение параметра оболочки `nocasematch`. Попытка воспользоваться неустановленной переменной с помощью подстановок **#**, **##**, **%**, **%%**, **//**, **/#**, **/%**, **^**, **^^**, **,** и **,,** в неинтерактивных оболочках, настраиваемых по команде `set -u`, приводит к выходу из оболочки.

Рассмотрим в качестве примера следующую подстановку переменной с текстом, заключенным в одиночные кавычки:

```
${переменная:=a'специальный_текст'b}
```

В данном примере *специальный_текст* распознается как заключаемый в кавычки. Но если в режиме работы по стандарту POSIX происходит подстановка переменной в двойных кавычках, то одиночные кавычки не определяют новый вложенный контекст заключения в кавычки. Впрочем, из этого правила имеются исключения: одиночные кавычки не обеспечивают контекст заключения в кавычки вместе с подстановками **#**, **##**, **%**, **%%**, **//**, **/#**, **/%**, **^**, **^^**, **,** и **,,**.

переменная=значение ...	Установить заданное ***значение*** в каждой указанной ***переменной***
${*переменная*}	Использовать значение указанной ***переменной***. Указывать фигурные скобки необязательно, если ***переменная*** отделяется от последующего текста. Но они требуются для обозначения переменных массивов
${*переменная*:-*значение*}	Использовать указанную ***переменную***, если она установлена, а иначе — заданное ***значение***
${*переменная*:=*значение*}	Использовать указанную ***переменную***, если она установлена, а иначе — заданное ***значение***, присвоив его указанной ***переменной***

`${`***переменная***`:?`***значение***`}`	Использовать указанную ***переменную***, если она установлена, в портивном случае вывести заданное ***значение*** и выйти из оболочки, если она неинтерактивная. Если же ***значение*** не задано, направить сообщение **"parameter null or not set"** (пустой или неустановленный параметр) в стандартный вывод ошибок **stderr**
`${`***переменная***`:+`***значение***`}`	Использовать заданное ***значение***, если указанная ***переменная*** установлена, а иначе — не использовать ничего
`${#`***переменная***`}`	Использовать длину указанной ***переменной***
`${#*}`, `${#@}`	Использовать количество позиционных параметров
`${`***переменная***`#`***шаблон***`}`	Использовать значение указанной ***переменной*** после удаления текста, совпадающего с заданным ***шаблоном*** слева. Удалить самый короткий совпадающий фрагмент
`${`***переменная***`##`***шаблон***`}`	То же, что и выше, но удалить самый длинный совпадающий фрагмент
`${`***переменная***`%`***шаблон***`}`	Использовать значение указанной ***переменной*** после удаления текста, совпадающего с заданным ***шаблоном*** справа. Удалить самый короткий совпадающий фрагмент
`${`***переменная***`%%`***шаблон***`}`	То же, что и выше, но удалить самый длинный совпадающий фрагмент
`${`***переменная***`^`***шаблон***`}`	Преобразовать значение указанной ***переменной*** в прописные буквы. Заданный ***шаблон*** вычисляется таким же образом, как и при совпадении с именами файлов. Если с шаблоном совпадает первая буква в значении указанной ***переменной***, то она преобразуется в прописную. Если же ***переменная*** указана как * или @, то видоизменяются позиционные параметры. А если ***переменная*** указана как массив с индексом * или @, то подстановка распространяется на все элементы массива
`${`***переменная***`^^`***шаблон***`}`	То же, что и выше, но с шаблоном сопоставляется каждая буква в строке

`${`***переменная,шаблон***`}`	То же, что и `${`***переменная^шаблон***`}`, но совпадающие буквы преобразуются в строчные. Применяется только к первому символу в строке
`${`***переменная,,шаблон***`}`	То же, что и выше, но с шаблоном сопоставляется каждая буква в строке
`${`***переменная***`@a}`	Использовать, как и в команде **declare**, значения признаков, представляющие атрибуты указанной ***переменной***, которая может быть массивом с индексом * или @, и тогда преобразование распространяется на все элементы массива
`${`***переменная***`@A}`	Строка в форме команды или операция присваивания, при вычислении которой воссоздается указанная ***переменная*** и ее атрибуты. Указанная ***переменная*** может быть массивом с индексом * или @, и тогда преобразование распространяется на все элементы массива
`${`***переменная***`@E}`	Вычислить значение указанной ***переменной*** с управляющими последовательностями в контексте $'...' (см. выше раздел "Управляющие последовательности символов"). Указанная ***переменная*** может быть массивом с индексом * или @, и тогда преобразование распространяется на все элементы массива
`${`***переменная***`@P}`	Вычислить значение указанной ***переменной*** с управляющими последовательностями в строке приглашения (см. ниже раздел "Специальные строки приглашений"). Указанная ***переменная*** может быть массивом с индексом * или @, и тогда преобразование распространяется на все элементы массива
`${`***переменная***`@Q}`	Заключить в кавычки значение указанной ***переменной***, чтобы вводить значения как входные данные. Указанная ***переменная*** может быть массивом с индексом * или @, и тогда преобразование распространяется на все элементы массива

`${!`***префикс****`}`, `${!`***префикс***`@}`	Перечислить переменные, имена которых начинаются с указанного ***префикса***
`${`***переменная***`:`***позиция***`}`, `${`***переменная***`:`***позиция***`:`***длина***`}`	Извлечь количество символов заданной ***длины***, начиная с обозначенной ***позиции***, отсчитываемой от нуля в указанной ***переменной***, а если ***длина*** не задана, то извлечь остальную часть строки. Заданные ***позиция*** и ***длина*** могут быть арифметическими выражениями. Отрицательная ***длина*** отсчитывается от конца строки. Если ***переменная*** указана как `*` или `@`, то подстановка выполняется по позиционным параметрам. Если задана нулевая ***позиция***, то в итоговый список включается позиционный параметр `$0`. Аналогично ***переменная*** может быть указана как массив с индексом `*` или `@`
`${`***переменная***`/`***шаблон***`/`***замена***`}`	Использовать значение указанной ***переменной***, заменив первое совпадение с заданным ***шаблоном*** обозначенной ***заменой***
`${`***переменная***`/`***шаблон***`}`	Использовать значение указанной ***переменной***, удалив первое совпадение с заданным ***шаблоном***
`${`***переменная***`//`***шаблон***`/`***замена***`}`	Использовать значение указанной ***переменной***, заменив каждое совпадение с заданным ***шаблоном*** обозначенной ***заменой***
`${`***переменная***`/#`***шаблон***`/`***замена***`}`	Использовать значение указанной ***переменной***, заменив совпадение с заданным ***шаблоном*** обозначенной ***заменой***. Совпадение должно произойти в начале значения указанной ***переменной***
`${`***переменная***`/%`***шаблон***`/`***замена***`}`	Использовать значение указанной ***переменной***, заменив совпадение с заданным ***шаблоном*** обозначенной ***заменой***. Совпадение должно произойти в конце значения указанной ***переменной***
`${!`***переменная***`}`	Использовать значение указанной ***переменной*** в качестве имени другой переменной, значение которой должно быть использовано (косвенная ссылка)

Примеры

```
$ u=up d=down blank=          # Присвоить значения
                              # трем переменным
                              # (последней — пустое значение)
$ echo ${u}root               # Здесь требуются фигурные
                              # скобки
uproot
$ echo ${u-$d}                # Отобразить значение
                              # переменной u
up                            # или d; переменная u
установлена,
                              # поэтому вывести ее значение
$ echo ${tmp-`date`}          # Если переменная tmp
                              # не установлена,
Tue Feb 2 22:52:57 EST 2016   # выполнить команду date
$ echo ${blank="no data"}     # Переменная blank установлена,
                              # поэтому вывести ее значение
                              # (пустую строку)
$ echo ${blank:="no data"}    # Переменная blank
                              # установлена, но
no data                       # пустая, поэтому выводится
                              # заданная символьная строка
$ echo $blank                 # Теперь переменная blank имеет
no data                       # новое значение

# Взять имя текущего каталога и удалить из него
# самую длинную строку, оканчивающуюся знаком /.
# В итоге удаляется начальная часть пути и остается
# лишь его конечная часть
$ tail=${PWD##*/}

# Использовать известное слово
$ word=supercalifragilisticexpialidocious

# Сменить регистр первой буквы
$ echo ${word^[r-t]}
Supercalifragilisticexpialidocious

# Сменить регистр всех совпадающих букв
$ echo ${word^^[r-t]}
SupeRcalifRagiliSTicexpialidociouS
```

Косвенные переменные

Косвенные переменные, иначе называемые *ссылками* на косвенные переменные (*namerefs*), — это переменные, именующие другие переменные. Все действия (ссылки, присваивания и изменения атрибутов), применяемые к косвенной переменной, выполняются над переменной, имя которой обозначает значение косвенной переменной. Косвенные переменные создаются по команде `declare -n`, удаляются по команде `unset -n`, а проверяются по команде `test -R`. Например:

```
$ greeting="hello, world"         # Обычное присваивание
                                  # значения переменной
$ declare -n message=greeting     # Объявить косвенную
                                  # переменную
$ echo $message                   # Получить через нее доступ к
hello, world                      # переменной greeting и
                                  # отобразить его
$ message="bye now"               # Присвоить через косвенную
                                  # переменную message значение
                                  # переменной greeting
$ echo $greeting                  # Продемонстрировать изменения
bye now                           # в содержимом переменной
                                  # greeting
```

В оболочке Bash предоставляется также специальный синтаксис, позволяющий одной переменной косвенно ссылаться на другую переменную, но присваивать значения переменным этот синтаксис не позволяет. В приведенном ниже примере демонстрируется его применение.

```
$ text=greeting          # Обычное присваивание значения
                         # одной переменной другой переменной
$ echo ${!text}          # Применение псевдонима
bye now
```

Если косвенная переменная употребляется в качестве управляющей циклом `for`, элементы такого цикла интерпретируются как имена переменных, на которые поочередно ссылается косвенная переменная:

```
$ declare -n nr             # Установить косвенную переменную
$ i=1                       # Организовать простой счетчик
```

```
$ for nr in v1 v2 v3        # Начать цикл
> do
>     nr=$((i++))           # Каждая переменная получает
                            # особое значение
> done
$ echo $v1 $v2 $v3          # Показать результаты
1 2 3
```

Преобразование существующей переменной в косвенную отменяет действие атрибутов **-c**, **-i**, **-l** и **-u** (описание команды `declare` см. ниже, в разделе "Встроенные команды").

Переменные, встроенные в оболочку

Встроенные переменные устанавливаются в оболочке автоматически. Как правило, они применяются в сценариях оболочки. Ко встроенным переменным применимы описанные выше шаблоны подстановки. Следует, однако, иметь в виду, что знак **$** не является частью имени переменной, хотя ссылка на нее всегда делается с помощью именно этого знака. Ниже перечислены встроенные переменные, доступные в любой оболочке, совместимой с Bourne shell.

Переменная	Что обозначает
$#	Количество аргументов в командной строке
$-	Действующие в настоящий момент параметры, предоставляемые в командной строке или задаваемые в команде **set**. Некоторые параметры устанавливаются в оболочке автоматически
$?	Выходное значение последней выполнявшейся команды
$$	Номер процесса, присвоенный оболочке
$!	Номер процесса, присвоенный последней команде, выполнявшейся в фоновом режиме
$0	Первое слово, т.е. имя команды. Будет иметь полный путь, если команда была найдена с помощью переменной окружения **$PATH**.
$*n*	Отдельные аргументы командной строки (позиционные параметры). В оболочке Bourne shell допускается непосредственно ссылаться только на девять позиционных параметров (***n*=1-9**). А в оболочке Bash допускается указывать количество позиционных параметров ***n*** больше **9**, если для этого используется форма **${*n*}**

Переменная	Что обозначает
$*, $@	Все аргументы командной строки (**$1**, **$2** и т.д.)
"$*"	Все аргументы командной строки в одной символьной строке ("**$1 $2**..."). Значения разделяются первым символом, указанным в переменной **$IFS**
"$@"	Все аргументы командной строки, заключенные по отдельности в кавычки ("**$1**" "**$2**" ...)

В оболочке Bash автоматически устанавливаются следующие дополнительные переменные[2].

Переменная	Что обозначает
$_	Временная переменная. Инициализируется путем к выполняемому сценарию или программе. В дальнейшем хранит последний аргумент предыдущей команды, а также имя файла **MAIL**, совпадающего при проверке почты
BASH	Полный путь для вызова данного экземпляра оболочки Bash
BASHOPTS	Доступный только для чтения список активизированных в настоящий момент параметров оболочки, разделяемый запятыми. Каждый элемент такого списка содержит достоверный параметр для команды **shopt -s**. Если данная переменная существует в рабочей среде при запуске оболочки Bash, обозначенные параметры устанавливаются в ней, прежде чем выполнять любые файлы запуска
BASHPID	Идентификатор текущего процесса, присвоенный оболочке Bash. Иногда значение этой переменной может отличаться от значения переменной **$$**
BASH_ALIASES	Переменная ассоциативного массива, каждый элемент которого содержит псевдоним, определенный по команде **alias**. При вводе элемента в этот массив создается новый псевдоним
BASH_ARGC	Переменная массива, каждый элемент которого содержит аргументы для соответствующей функции или сценария, вызываемого по команде-точке (**.**). Устанавливается только в расширенном режиме отладки по команде **shopt -s extdebug**. В исходное состояние эту переменную установить нельзя

[2]Не все эти переменные устанавливаются всегда. Например, переменные COMP* получают свои значения только при выполнении функций автозавершения.

Переменная	Что обозначает
BASH_ARGV	Переменная массива, аналогичная переменной **BASH_ARGC**. Каждый элемент массива содержит аргумент, передаваемый функции или сценарию, вызываемому по команде-точке (**.**). Действует подобно стеку, куда помещаются значения при каждом вызове. Таким образом, последний элемент содержит последний аргумент самой последней вызывавшейся функции или сценария. Устанавливается только в расширенном режиме отладки по команде **shopt -s extdebug**. В исходное состояние эту переменную установить нельзя
BASH_CMDS	Переменная ассоциативного массива, каждый элемент которого содержит ссылку на команду во внутренней хеш-таблице, которая ведется по команде **hash**. В качестве индекса служит имя команды, а в качестве значения — полный путь к команде. При вводе нового элемента в данный массив соответствующая команда вводится в хеш-таблицу
BASH_COMMAND	Команда, которая выполняется в текущий момент или готовится к выполнению. В обработчике прерываний это команда, выполняемая при появлении прерывания
BASH_EXECUTION_STRING	Строковый аргумент, передаваемый параметру **-c**
BASH_LINENO	Переменная массива, действующая вместе с соответствующими переменными **BASH_SOURCE** и **FUNCNAME**. Так, для каждого заданного номера функции **i**, начинающегося с нуля, в подстановке **${BASH_LINENO[i]}** может быть сделан вызов функции **${FUNCNAME[i]}** из файла **${BASH_SOURCE[i]}**. Полученная информация сохраняется вместе с вызовом самой последней функции. В исходное состояние эту переменную установить нельзя
BASH_REMATCH	Переменная массива, значение которой присваивается с помощью операции **=~** в конструкции **[[ ]]**. По нулевому индексу содержится текст, полностью совпадающий с шаблоном, а по остальным индексам — текст, совпадающий с подвыражениями в круглых скобках. Эта переменная доступна только для чтения

Переменная	Что обозначает
BASH_SOURCE	Переменная массива, содержащая имена исходных файлов. Каждый элемент соответствует элементам массивов в переменных **FUNCNAME** и **BASH_LINENO**. В исходное состояние эту переменную установить нельзя
BASH_SUBSHELL	Значение этой переменной инкрементируется на единицу всякий раз, когда создается подоболочка или рабочая среда
BASH_VERSINFO[0]	Основной номер версии или выпуска оболочки Bash
BASH_VERSINFO[1]	Дополнительный номер версии или выпуска оболочки Bash
BASH_VERSINFO[2]	Номер версии исправлений
BASH_VERSINFO[3]	Номер версии сборки
BASH_VERSINFO[4]	Состояние выпуска
BASH_VERSINFO[5]	Тип машины. То же, что и переменная **$MACHTYPE**
BASH_VERSION	Символьная строка, описывающая версию оболочки Bash
COMP_CWORD	Служит для автозавершения. Содержит индекс массива в переменной **COMP_WORDS**, указывающий текущее положение курсора
COMP_KEY	Служит для автозавершения. Содержит текущий или последний ключ в последовательности, обусловившей вызов текущей функции автозавершения
COMP_LINE	Служит для автозавершения. Содержит текущую командную строку
COMP_POINT	Служит для автозавершения. Содержит положение курсора в виде символьного индекса массива в переменной **$COMP_LINE**
COMP_TYPE	Служит для автозавершения. Содержит символ, описывающий тип автозавершения. Для нормального завершения — это один из знаков табуляции; для списков завершения после двух знаков табуляции — знак **?**; для списка альтернативных вариантов завершения частично введенного слова — знак **!**; для завершения измененного слова — знак **@**; а для меню вариантов завершения — знак **%**

Переменная	Что обозначает
COMP_ WORDBREAKS	Служит для автозавершения. Содержит символы, интерпретируемые в библиотеке **readline** как разделители слов при автозавершении последних
COMP_WORDS	Служит для автозавершения. Содержит массив отдельных слов из командной строки
COPROC	Переменная массива, содержащего дескрипторы файлов, применяемые для взаимодействия с неименованными сопроцессами. Подробнее об этом см. далее, в разделе "Сопроцессы"
DIRSTACK	Переменная массива, содержащего стек каталогов, отображаемый по команде **dirs**. При изменении существующих элементов этого массива изменяется и стек каталогов, но помещать в него элементы и извлекать их из него можно лишь по соответствующим командам **pushd** и **popd**
EUID	Переменная, доступная только для чтения и содержащая числовой действующий идентификатор текущего пользователя
FUNCNAME	Переменная массива, содержащего имена функций. Каждый элемент этого массива соответствует элементам массивов в переменных **BASH_SOURCE** и **BASH_LINENO**
FUNCNEST	Значение этой переменной, большее нуля, определяет максимальный уровень вложения вызовов функций. Если этот уровень превышен, выполнение текущей команды следует прервать
GROUPS	Переменная массива, содержащего список числовых идентификаторов групп, членом которых является текущий пользователь
HISTCMD	Номер текущей команды в предыстории
HOSTNAME	Номер текущего хоста (т.е. сетевого узла)
HOSTTYPE	Символьная строка, описывающая главную систему
LINENO	Номер текущей строки кода в сценарии или функции
MACHTYPE	Символьная строка, описывающая главную систему в формате ***ЦП-организация-система*** по общей лицензии GNU
MAPFILE	Используемый по умолчанию массив команд **mapfile** и **readarray**. Подробнее об этом см. далее описание команды **mapfile** в разделе "Встроенные команды"

Переменная	Что обозначает
`OLDPWD`	Предыдущий рабочий каталог, устанавливаемый по команде `cd` или наследуемый из рабочей среды, если этот каталог именуется в ней
`OPTARG`	Значение аргумента для последнего параметра, обработанного командой `getopts`
`OPTIND`	Числовой индекс значения переменной `OPTARG`
`OSTYPE`	Символьная строка, описывающая операционную систему
`PIPESTATUS`	Переменная массива, содержащего коды завершения команд в самом последнем конвейере приоритетного режима. Но следует иметь в виду, что конвейер может содержать только одну команду
`PPID`	Идентификационный номер процесса, присвоенный оболочке, являющейся родительской для текущей оболочки
`PWD`	Текущий рабочий каталог, устанавливаемый по команде `cd`
`RANDOM[=n]`	Новое случайное число, формируемое по каждой ссылке на эту переменную, начиная с целого числа *n*, если оно задано
`READLINE_LINE`	Служит для применения вместе с командой `bind -x`. В этой переменной доступно содержимое буфера редактирования
`READLINE_POINT`	Служит для применения вместе с командой `bind -x`. Содержит индекс значения переменной `READLINE_LINE` в точке вставки
`REPLY`	Ответ по умолчанию. Применяется в цикле `select` и команде `read`
`SECONDS[=n]`	Количество секунд, прошедших с момента запуска оболочки, или же количество секунд, прошедших с момента присваивания значения этой переменной, плюс *n*, если *n* задано
`SHELLOPTS`	Доступный только для чтения, разделяемый запятыми список параметров оболочки (для команды `set -o`). Если эта переменная устанавливается в рабочей среде при запуске оболочки Bash, последняя активизирует каждый параметр, имеющийся в данном списке, прежде чем читать любые файлы запуска
`SHLVL`	Значение этой переменной всякий раз инкрементируется на единицу, когда запускается оболочка Bash

Переменная	Что обозначает
UID	Переменная, доступная только для чтения и содержащая числовой реальный идентификатор текущего пользователя

Многие из перечисленных выше переменных обеспечивают поддержку режима автозавершения (см. далее раздел “Автозавершение вводимых команд”) или отладки в оболочке Bash (подробнее см. по адресу `http://bashdb.sourceforge.net`).

Другие переменные оболочки

Перечисленные ниже переменные не устанавливаются в оболочке автоматически, хотя многие из них способны оказывать влияние на режим работы оболочки Bash. Как правило, они устанавливаются в файле `.bash_profile` или `.profile`, где их можно определить в соответствии со своими потребностями. Этим переменным можно присвоить значения, выдав команды в следующей форме:

переменная=значение

В приведенном ниже списке переменных предполагается указание типа значения при определении этих переменных.

Переменная	Что обозначает
BASH_COMPAT	Если в этой переменной установлено десятичное или целое значение (например, **4.3** или **43**), соответствующее поддерживаемому в оболочке уровню совместимости, то активизируется заданный уровень совместимости (например, уровни **4.3** и **43** соответствуют команде **shopt -s compat43**). Если же эта переменная установлена в исходное состояние или в ней задана пустая символьная строка, то устанавливается уровень совместимости, присущий текущей оболочке. Команда **shopt** не изменяет значение этой переменной, хотя она может наследоваться из рабочей среды

Переменная	Что обозначает
BASH_ENV	Если эта переменная устанавливается при запуске оболочки, она именует файл, обрабатываемый для команд инициализации. Значение этой переменной подвергается подстановке параметров, команд и арифметической подстановке, прежде чем интерпретировать его как имя файла
BASH_LOADABLES_PATH	Один или несколько разделяемых двоеточием путей, по которым осуществляется поиск динамически загружаемых встроенных команд, указанных по команде **enable**
BASH_XTRACEFD=*n*	Дескриптор файла, в который оболочка Bash записывает результат трассировки, выводимый из команды **set -x**
CDPATH=*каталоги*	Каталоги, искомые по команде **cd**. Допускаются сокращения в смене каталогов. Эта переменная устанавливается в исходное состояние по умолчанию
CHILD_MAX=*n*	Максимально устанавливаемое количество процессов, для которых оболочка запоминает коды завершения. Максимальное значение этой переменной равно **8192**, а минимальное зависит от конкретной системы
COLUMNS=*n*	Ширина столбца на экране. Применяется в режимах редактирования строк и списках, формируемых в цикле **select**. По умолчанию выбирается ширина столбца на текущем терминале
COMPREPLY=(*слова* ...)	Переменная массива, из которого оболочка Bash читает возможные варианты, формируемые функцией автозавершения
EMACS	Если значение этой переменной начинается с буквы **t**, то оболочка Bash посчитает, что она выполняется в буфере редактора Emacs, отменяя редактирование строк
ENV=*файл*	Наименование сценария, выполняемого при запуске в режиме работы по стандарту POSIX или же в том случае, если оболочка Bash вызывается по пути **/bin/sh**. Это удобно для хранения определений псевдонимов и функций. Например: **ENV=$HOME/.shellrc**
EXECIGNORE=*список_шаблонов*	Разделяемый двоеточиями список глобальных шаблонов, описывающих ряд имен файлов, игнорируемых при поиске исполняемых файлов. Это удобно для игнорирования общих библиотечных файлов с правами доступа на исполнение. При этом во внимание принимается значение параметра оболочки **extglob**

Переменная	Что обозначает
`FCEDIT=`*`файл`*	Редактор, применяемый в команде `fc`. По умолчанию выбирается редактор, доступный по пути `/bin/ed`, когда оболочка Bash работает в режиме по стандарту POSIX. В противном случае по умолчанию выбирается редактор, устанавливаемый в переменной `$EDITOR`, а если эта переменна не установлена — редактор `vi`
`FIGNORE=`*`список_шаблонов`*	Разделяемый двоеточиями список суффиксов, описывающих ряд имен файлов, игнорируемых при автозавершении имен файлов средствами библиотеки `readline`
`GLOBIGNORE=`*`список_шаблонов`*	Разделяемый двоеточиями список шаблонов, описывающий ряд имен файлов, игнорируемых при сопоставлении с шаблоном. При этом во внимание принимаются значения параметров оболочки `nocasematch` и `extglob`
`HISTCONTROL=`*`список`*	Разделяемый двоеточиями список значений, определяющих порядок сохранения команд в файле предыстории. В этом списке распознаются значения признаков `ignoredups`, `ignorespace`, `ignoreboth` и `erasedups`
`HISTFILE=`*`файл`*	Файл, в котором хранится предыстория выполнения команд. По умолчанию выбирается файл `~/.bash_history`
`HISTFILESIZE=`*`n`*	Количество строк, сохраняемых в файле предыстории. Это количество может отличаться от количества сохраняемых команд. Если оно равно нулю, то ни одна из команд не сохраняется. А если оно отрицательное или нечисловое, то никаких ограничений не накладывается. По умолчанию выбирается количество сохраняемых строк, равное `500`
`HISTIGNORE=`*`список`*	Разделяемый двоеточиями список шаблонов, с которыми должна полностью совпадать командная строка. Совпадающие строки *не* сохраняются в файле предыстории. Неэкранированный знак `&` в шаблоне обозначает совпадение с предыдущей строкой в предыстории. При этом значение параметра оболочки `extglob` игнорируется
`HISTSIZE=`*`n`*	Количество команд, сохраняемых в файле предыстории. Если оно равно нулю, то ни одна из команд не сохраняется. А если оно отрицательное или нечисловое, то никаких ограничений не накладывается. По умолчанию выбирается количество сохраняемых команд, равное `500`

Переменная	Что обозначает
HISTTIMEFORMAT=*строка*	Форматирующая строка (см. оперативную страницу руководства *strftime*(3)), предназначенная для вывода отметок времени вместе с командами из предыстории по команде **history**. Если эта строка установлена (даже пустой), оболочка Bash сохраняет отметки времени вместе с командами в файле предыстории
HOME=*каталог*	Начальный каталог, устанавливаемый по команде **login** (из файла **/etc/passwd**)
HOSTFILE=*файл*	Имя файла в том же самом формате, что и файл, доступный по пути **/etc/hosts**. Оболочка Bash должна использовать этот файл для поиска имен хостов в целях их автозавершения
IFS='*символы*'	Разделители полей ввода. По умолчанию выбираются знаки пробела, табуляции и новой строки
IGNOREEOF=*n*	Числовое значение, обозначающее количество последовательных знаков окончания файла, которые требуется ввести, прежде чем произойдет выход из оболочки. Если это значение оказывается пустым или нечисловым, то по умолчанию выбирается значение 10. Применяется только в интерактивных оболочках
INPUTRC=*файл*	Файл инициализации для библиотеки **readline**. Заменяет выбираемый по умолчанию файл **~/.inputrc**
LANG=*языковый_стандарт*	Выбираемый по умолчанию языковый стандарт. Используется, если не установлена ни одна из перечисленных ниже переменных **LC_***
LC_ALL=*языковый_стандарт*	Текущий языковый стандарт. Переопределяет значения переменной **LANG** и других переменных **LC_***
LC_COLLATE=*языковый_стандарт*	Языковой стандарт, применяемый при сортировке символов в заданном порядке
LC_CTYPE=*языковый_стандарт*	Языковой стандарт, применяемый в функциях классов символов (см. выше раздел "Метасимволы подстановки имен файлов")
LC_MESSAGES=*языковый_стандарт*	Языковой стандарт, применяемый для преобразования символьных строк в форме $"..."
LC_NUMERIC=*языковый_стандарт*	Языковой стандарт, применяемый для представления символа, обозначающего десятичную точку
LC_TIME=*языковый_стандарт*	Языковой стандарт, применяемый в форматах даты и времени

Переменная	Что обозначает
LINES=*n*	Высота экрана. Применяется в списках, формируемых в цикле **select**. По умолчанию выбирается высота экрана на текущем терминале
MAIL=*файл*	Файл, выбираемый по умолчанию для проверки входящей почты. Устанавливается по команде **login**
MAILCHECK=*n*	Количество секунд, проходящих между последовательными проверками почты. По умолчанию выбирается значение **60** (т.е. одна минута)
MAILPATH=*файлы*	Один или несколько файлов, разделяемых двоеточиями и предназначенных для проверки входящей почты. Вместе с каждым файлом можно также указать дополнительное сообщение, выводимое оболочкой при увеличении размера файла. Сообщения отделяются от имен файлов знаком **?**, а по умолчанию выводится сообщение "**You have mail in $_**" (У вас есть почта в переменной **$_**), где имя временной переменной **$_** заменяется именем файла с поступившей почтой. Например, переменную **MAILPATH** можно установить следующим образом: **MAILPATH="$MAIL?Candy gram!:/etc/motd?New Login Message**
OPTERR=*n*	Если в этой переменной установлено значение **1** (по умолчанию), то оболочка Bash выводит сообщения об ошибках из встроенной команды **getopts**
PATH=*список_каталогов*	Один или больше разделяемых двоеточием путей, по которым осуществляется поиск команд для последующего выполнения. По умолчанию выбирается составленный путь **usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/bin:/sbin:.**. Во многих системах по умолчанию выбирается путь **/bin:/usr/bin**
POSIXLY_CORRECT=*строка*	Если эта переменная устанавливается при запуске или в ходе выполнения оболочки Bash, последняя переходит в режим работы по стандарту POSIX, отменяя режим работы и видоизменяя средства, вступающие в конфликт со стандартом POSIX
PROMPT_COMMAND=*команда*	Если эта переменная установлена, оболочка Bash выполняет заданную команду всякий раз, прежде чем выводить основное приглашение

Переменная	Что обозначает
PROMPT_DIRTRIM=*n*	Обозначает, сколько завершающих составляющих каталогов остается для специальной строки приглашения **\w** или **\W** (см. далее раздел "Специальные строки приглашений"). Удаляемые составляющие каталогов заменяются многоточиями
PS0=*строка*	Символьная строка, выводимая интерактивными оболочками после чтения команды, но перед ее выполнением
PS1=*строка*	Основная строка приглашения. По умолчанию выбирается строка **'\s-\v\$ '**
PS2=*строка*	Вспомогательное приглашение, применяемое в многострочных командах. По умолчанию выбирается приглашение **>**
PS3=*строка*	Строка приглашения в циклах **select**. По умолчанию выбирается строка приглашения **#?**
PS4=*строка*	Строка приглашения к трассировке выполняемых команд (**bash -x** или **set -x**). По умолчанию выбирается приглашение **+**. Оболочки, выполняемые как корневые, не наследуют эту переменную из рабочей среды
SHELL=*файл*	Наименование оболочки, выбираемой пользователем по умолчанию (например, по пути **/bin/sh**). Оболочка Bash устанавливает эту переменную, если при запуске она отсутствует в рабочей среде
TERM=*строка*	Тип терминала
TIMEFORMAT=*строка*	Форматирующая строка для вывода времени с помощью ключевого слова **time**. Подробнее об этом см. оперативную страницу руководства *bash*(1)
TMOUT=*n*	Если никакой команды не введено по истечении ***n*** секунд, то происходит выход из оболочки. Значение, устанавливаемое в этой переменной, оказывает также влияние на действие команды **read** и цикла **select**
TMPDIR=*каталог*	В указанном ***каталоге*** размещаются временно создаваемые файлы, применяемые в оболочке
auto_ resume=*список*	Допускает применение простых символьных строк для возобновления приостановленных заданий. Если установлено значение **exact**, символьная строка должна точно совпадать с именем команды. А если установлено значение **substring**, то символьная строка может совпадать с подстрокой из имени команды

Переменная	Что обозначает
histchars=*символы*	Два или три символа, определяющие порядок подстановки предыстории в стиле оболочки **csh**. Первый символ сигнализирует о событии в предыстории команд, второй символ обозначает "быструю" подстановку, а третий — начало комментария. По умолчанию для этой цели выбираются символы **!^#**. См. также раздел "Предыстория команд в стиле оболочки C Shell"

Массивы

В оболочке Bash предоставляются два вида массивов: *индексированные массивы*, где индексами являются целые числа от нуля и выше, а также *ассоциативные массивы*, где индексами являются символьные строки.

Индексированные массивы

В оболочке Bash поддерживаются одномерные массивы. Первый элемент такого массива нумеруется нулевым. Никаких ограничений на количество элементов массива в оболочке Bash не накладывается. Массивы инициализируются с помощью специальной формы присваивания, как показано в приведенном ниже примере.

```
message=(hi there how are you today)
```

Задаваемые значения становятся элементами массива. Значения могут быть присвоены отдельным элементам массива следующим образом:

```
message[0]=hi                  # Это нелегкий путь
message[1]=there
message[2]=how
message[3]=are
message[4]=you
message[5]=today
```

Объявлять индексные массивы не требуется. Массив можно создать по любой достоверной ссылке на индексированную переменную.

Для ссылки на массивы служит синтаксическая форма `${...}`. Хотя она не требуется для ссылки на массивы в форме `((...))` команды `let`, которая автоматически заключает в кавычки указанные элементы. Следует, однако, иметь в виду, что квадратные скобки **[** и **]** вводятся буквально, т.е. они не означают какой-то дополнительный синтаксис.

Отрицательные индексы отсчитываются от последнего индекса с прибавлением единицы. Ниже приведены некоторые примеры применения индексированных массивов.

```
$ a=(0 1 2 3 4 5 6 7 8)   # Создать индексированный массив
$ echo ${a[4]}            # Использовать положительный индекс
4
$ echo ${a[-2]}           # Использовать индекс: 8 + 1 - 2 = 7
7
```

Подстановки массивов

Подстановки переменных для массивов и их элементов выполняются в следующих формах.

${*имя*[*i*]}	Использовать элемент ***i*** массива ***имя***, где ***i*** может быть арифметическим выражением, как поясняется ниже, в разделе "Арифметические выражения"
${*имя*}	Использовать нулевой элемент массива ***имя***
${*имя*[*]}**, **${*имя*[@]}	Использовать все элементы массива ***имя***
${#*имя*[*]}**, **${#*имя*[@]}	Использовать ряд элементов массива ***имя***

Ассоциативные массивы

В оболочке Bash поддерживаются также ассоциативные массивы, где индексами являются символьные строки, а не числа, как в команде `awk`, реализующей одноименный язык. В данном случае квадратные скобки (**[** и **]**) действуют подобно двойным кавычкам. Ассоциативные массивы должны быть объявлены с помощью параметра **-A** в командах `declare`, `local` и `readonly`. Специальный синтаксис позволяет присваивать значения сразу нескольким элементам ассоциативного массива, как показано ниже.

```
data=([joe]=30 [mary]=25)       # Присваивание значений
                                # элементам ассоциативного
                                # массива
message=([0]=hi [2]=there)      # Присваивание значений
                                # элементам индексированного
                                # массива
```

Для извлечения значений из массива служат формы `${data[joe]}` и `${data[mary]}`. Специальные выражения для извлечения всех индексов из ассоциативного массива действуют таким же образом, как и в индексированных массивах.

Специальные строки приглашений

Значения переменных `PS0`, `PS1`, `PS2` и `PS4` обрабатываются в оболочке Bash, принимая во внимание следующие управляющие последовательности символов.

`\a`	Звонок (BEL в коде ASCII) — звуковой или визуальный предупреждающий сигнал (восьмеричное значение **`07`**)
`\A`	Текущее время в 24-часовом формате ЧЧ:ММ
`\d`	Дата в формате "день недели месяц день"
`\D{`*`формат`*`}`	Дата, указанная в заданном ***формате*** (см. оперативную страницу руководства *strftime*(3)). Указывать фигурные скобки необязательно
`\e`	Символ перехода (Escape в коде ASCII; восьмеричное значение **`033`**)
`\h`	Имя хоста (т.е. сетевого узла) вплоть до первой точки
`\H`	Полное имя хоста
`\j`	Текущее количество заданий
`\l`	Базовое имя терминального устройства в оболочке
`\n`	Символ новой строки
`\r`	Символ перевода каретки
`\s`	Имя оболочки (базовое имя в переменной **`$0`**)
`\t`	Текущее время в 24-часовом формате ЧЧ:ММ:СС
`\T`	Текущее время в 12-часовом формате ЧЧ:ММ:СС
`\u`	Имя текущего пользователя
`\v`	Текущая версия оболочки Bash
`\V`	Текущий выпуск (версия плюс версия исправлений) оболочки Bash

`\W`	Базовое имя текущего каталога, где обозначение начального каталога **`$HOME`** сокращается до **~**. См. также приведенное ранее описание переменной **`PROMPT_DIRTRIM`**
`\!`	Номер строки с данной командой, сохраненной в предыстории
`\#`	Номер данной команды (в соответствии с подсчетом команд, который ведется в текущей оболочке)
`\$`	Если действующий идентификатор пользователя равен нулю, то символ **#**, а иначе — символ **$**
`\@`	Текущее время в 12-часовом формате a.m./p.m. (т.е. до и после полудня)
`\nnn`	Символ, представленный восьмеричным значением ***nnn***
`\\`	Символ обратной косой черты
`\[`	Начало последовательности непечатаемых символов, например, для выделения или изменения цвета в эмуляторе терминала
`\]`	Конец последовательности непечатаемых символов

Переменные **PS0**, **PS1**, **PS2** и **PS4** подвергаются подстановке управляющих последовательностей символов, переменных, команд и арифметических выражений. Сначала обрабатываются управляющие последовательности символов, затем выполняются подстановки, если параметр оболочки `promptvars` активизирован по команде `shopt` (по умолчанию).

В режиме работы по стандарту POSIX дело обстоит несколько иначе. Значения переменных `PS1` и `PS2` подвергаются подстановке параметров, знак `!` заменяется номером строки с текущей командой в предыстории, а знаки `!!` — собственно знаком восклицания.

Арифметические выражения

Команда `let` выполняет целочисленные арифметические операции. Оболочка Bash обеспечивает порядок подстановки значений арифметических выражений (для применения в качестве аргументов или сохранения в переменных), а также базовые преобразования, если таковые требуются. Ниже перечислены формы представления арифметических выражений.

`$(( выражение ))`	Использовать значение выражения, указанного в круглых скобках. Оболочка Bash попытается произвести синтаксический анализ формы `$((...))` как арифметического выражения, прежде чем анализировать ее как вложенную подстановку команд
`B#n`	Интерпретировать заданное целочисленное значение *`n`* по указанному основанию *`B`*. Например, форма `8#100` обозначает восьмеричный эквивалент десятичного числа `64`

Операции

В оболочке Bash поддерживаются арифметические операции, перечисленные ниже в порядке их старшинства. Большая их часть унаследована из языка программирования C.

Операция	Описание
`++ --`	Автоматическое инкрементирование и декрементирование (как префиксное, так и постфиксное)
`+ -`	Унарный плюс и минус
`! ~`	Логическое отрицание и двоичная инверсия (дополнение до единицы или получение обратного кода)
`**`	Возведение в степень
`* / %`	Умножение, деление, взятие модуля (получение остатка от деления)
`+ -`	Сложение и вычитание
`<< >>`	Поразрядный сдвиг влево и вправо
`< <= > >=`	Операции сравнения: меньше, меньше или равно, больше, больше или равно
`== !=`	Операции сравнения: равно, неравно (обе операции выполняются слева направо)
`&`	Логическая операция поразрядное И
`^`	Логическая операция поразрядное исключающее ИЛИ
`|`	Логическая операция поразрядное ИЛИ
`&&`	Логическая операция И (укороченная)
`||`	Логическая операция ИЛИ (укороченная)
`?:`	Встроенное вычисление по условию

Операция	Описание
`= += -=`	Присваивание
`*= /= %=`	—"—
`<<= >>=`	—"—
`&= ^= |=`	—"—
`,`	Последовательное вычисление выражений

Примечания

Команда `let` и форма `((...))` встроены в оболочку, и поэтому в них доступны значения переменных. Имя переменной совсем не обязательно предварять знаком денежной единицы (**$**), чтобы извлечь ее значение, хотя и такой способ обращения к переменной вполне работоспособен.

Команда `let` возвращает не вполне очевидный код завершения. Так, нулевой код свидетельствует об удачном завершении данной команды и ненулевом математическом результате, а ненулевой код — о неудачном завершении и нулевом математическом результате.

Примеры

```
let "count=0" "i = i + 1"       # Присвоить значения
                                # переменным i и count
let "num % 2"                   # Завершить удачно, если
                                # переменная num содержит
                                # нечетное число
(( percent <= 0 &&              # Проверить пределы значения
   percent <= 100 ))            # переменной percent
a=5 b=2                         # Задать ряд значений
echo $(("a" + "b"))             # Переменные можно заключать
                                # в двойные кавычки
```

Дополнительные сведения и примеры арифметических операций см. далее в описании команды `let`, приведенном в разделе “Встроенные команды”.

Предыстория выполнения команд

В оболочке Bash допускается отображать или видоизменять предыдущие команды. Используя команду `history`, можно вести список команд, сохраняемых оболочкой в предыстории. Подробнее об этом см. далее описание команды `history` в разделе "Встроенные команды". Все, а не только интерактивные оболочки, где активизируется предыстория выполнения команд (по команде `set -o history`), сохраняют такую предысторию.

В этом разделе основное внимание уделяется функциональным средствам редактирования команд, сохраняемых в предыстории их выполнения. Команды, хранящиеся в списке предыстории их выполнения, можно видоизменить, используя:

- режим редактирования строк;
- команду `fc`;
- предысторию в стиле оболочки C shell.

Режим редактирования строк

В режиме редактирования строк эмулируются многие функциональные средства текстовых редакторов `vi` и Emacs. В этом режиме список предыстории команд интерпретируется как файл. Как только текстовый редактор будет вызван, для перехода к строке с командой, которую требуется выполнить, нажимаются соответствующие клавиши редактирования. Строку с командой можно также изменить, прежде чем выполнять команду. И как только команда будет готова к выполнению, достаточно нажать клавишу <Enter>.

По умолчанию выбирается режим редактирования строк в текстовом редакторе Emacs. Для контроля над редактированием строк с командами следует воспользоваться командой `set -o vi` или `set -o emacs`. В обеих командах нельзя использовать переменные для указания текстового редактора.

Следует также иметь в виду, что режим редактирования строк в текстовом редакторе `vi` запускается в режиме ввода. Чтобы ввести команду `vi`, следует сначала нажать клавишу <Esc>.

Ниже перечислены наиболее употребительные клавиши редактирования.

Редактор `vi`	**Редактор Emacs**	**Результат**
<k>	<CTRL+p>	Получить предыдущую команду
<j>	<CTRL+n>	Получить следующую команду
</> ***строка***	<CTRL+n> ***строка***	Получить предыдущую команду, содержащую заданную ***строку***
<h>	<CTRL+b>	Переместить курсор на один символ назад
<l>	<CTRL+f>	Переместить курсор на один символ вперед
<b>	<Esc+b>	Переместить курсор на одно слово назад
<w>	<Esc+f>	Переместить курсор на одно слово вперед
<X>	<Del>	Удалить предыдущий символ
<x>	<Ctrl+d>	Удалить символ под курсором
<dw>	<Esc+d>	Удалить слово вперед
<db>	<Esc+h>	Удалить слово назад
<xp>	<Ctrl+t>	Переставить два символа

В режиме редактирования строк в обоих упомянутых выше текстовых редакторах допускается пользоваться клавишами управления курсором для перемещения по предыстории сохраненных команд.

Команда `fc`

Наименование команды `fc` сокращенно обозначает "команда поиска" или "команда исправления", поскольку она делает и то и другое. В частности, для обращения к списку команд в предыстории их выполнения служит команда `fc -l`, а для их редактирования — команда `fc -e`. Подробнее см. далее описание команды `fc` в разделе "Встроенные команды".

Примеры

```
$ history             # Перечислить последние 16 команд
$ fc -l 20 30         # Перечислить команды с 20-й по 30-ю
$ fc -l -5            # Перечислить последние 5 команд
$ fc -l cat           # Перечислить все команды с момента
                      # выполнения последней команды cat
$ fc -l 50            # Перечислить все команды с момента
                      # выполнения 50-й команды
$ fc -ln 5 > doit     # Сохранить 5-ю команду в файле doit
$ fc -e vi 5 20       # Отредактировать команды с 5-й по 20-ю
                      # в редакторе vi
$ fc -e emacs         # Отредактировать предыдущую команду
                      # в редакторе Emacs
```

Совет

Редактировать строки предыстории команд в диалоговом режиме проще, чем с помощью команды fc, поскольку в этом режиме можно легко перемещаться по списку команд в предыстории их выполнения, используя команды избранного текстового редактора, при условии, конечно, что им является редактор vi или Emacs! Для обхода предыстории команд можно также воспользоваться клавишами <↑> и <↓>.

Предыстория команд в стиле оболочки C shell

Помимо функциональных средств редактирования в диалоговом режиме и команды fc по стандарту POSIX, в оболочке Bash поддерживается также режим редактирования строк с командами, аналогичный принятому в оболочке Berkeley C shell (csh). Этот режим можно отменить по команде set +H. Многие пользователи отдают предпочтение функциональным средствам редактирования в диалоговом режиме, но данная возможность окажется удобной тем, кто привык пользоваться оболочкой csh.

В режиме работы по стандарту POSIX подстановка предыстории не происходит в двойных кавычках. Она всегда запрещается в одиночных кавычках. В версии Bash 5.0 команда set +H станет выполняться по умолчанию.

Обозначения событий

Обозначения событий помечают слово в командной строке как подстановку предыстории и перечислены ниже.

Обозначение	Описание
`!`	Начало подстановки предыстории
`!!`	Предыдущая команда
`!n`	Номер *n* команды в предыстории
`!-n`	*n*-я команда, отсчитываемая в предыстории обратно от текущей команды
`!строка`	Самая последняя команда, начинающаяся с заданной *строки*
`!?строка[?]`	Самая последняя команда, содержащая заданную *строку*
`#`	Строка с текущей командой до настоящего момента (совершенно бесполезно)
`^прежняя^новая^`	Быстрая подстановка; заменяет строку *прежняя* на строку *новая* в предыдущей команде и выполняет видоизмененную команду

Подстановка слов

Спецификаторы слов позволяют извлекать отдельные слова из строк с предыдущими командами. Они указываются после обозначения события и отделяются от него двоеточием. Указывать двоеточие необязательно, если далее приводится любой из следующих спецификаторов: `^`, `$`, `*`, `-` или `%`. Соответствующие спецификаторы слов перечислены ниже.

Спецификатор	Описание
`:0`	Наименование команды
`:n`	Аргумент номер *n*
`^`	Первый аргумент
`$`	Последний аргумент
`%`	Аргумент, совпадающий с критерием поиска `!?строка?`
`:n-m`	Аргументы от *n* до *m*
`-m`	Слова от `0` до *m*; то же, что и `0-m`
`:n-`	Аргументы от *n* до предпоследнего
`:n*`	Аргументы от *n* до последнего; то же, что и `n-$`
`*`	Все аргументы; то же, что и `^-$` или `1-$`

Модификаторы предыстории

Видоизменить подстановки слов и команд можно несколькими способами. Ниже перечислены модификаторы вывода, подстановки и заключения в кавычки.

Модификатор	Описание
`:p`	Отобразить, но не выполнять команду
`:s/`***прежняя***`/`***новая***	Подставить строку ***новая*** вместо строки ***прежняя***, но только первый экземпляр
`:gs/`***прежняя***`/`***новая***	Подставить строку ***новая*** вместо строки ***прежняя***, причем все экземпляры
`:as/`***прежняя***`/`***новая***	То же, что и модификатор `:gs`
`:Gs/`***прежняя***`/`***новая***	Аналогично модификатору `:gs`, но подстановка применяется ко всем словам в командной строке
`:&`	Повторить предыдущую подстановку (команду `:s` или `^`), но только первый экземпляр
`:g&`	Повторить предыдущую подстановку, причем все экземпляры
`:q`	Заключить список слов в кавычки
`:x`	Заключить в кавычки отдельные слова

Ниже перечислены модификаторы усечения.

Модификатор	Описание
`:r`	Извлечь первый доступный корень из пути (т.е. часть пути до последней точки)
`:e`	Извлечь первое доступное расширение пути (т.е. часть пути после последней точки)
`:h`	Извлечь первое доступное заглавие пути (т.е. часть пути до последней косой черты)
`:t`	Извлечь первое доступное окончание пути (т.е. часть пути после последней косой черты)

Автозавершение вводимых команд

В оболочке Bash и библиотеке `readline` предоставляются функциональные средства для *автозавершения* вводимых

команд, благодаря которым можно ввести часть наименования команды и нажать клавишу табуляции, а оболочка Bash частично или полностью заполнит оставшуюся часть наименования команды или имени файла. *Автозавершение* дает возможность программирующему на языке оболочки написать код для специальной настройки списка возможных вариантов завершения, который оболочка Bash предоставит в дальнейшем для конкретного частично введенного слова. Это достигается определенным сочетанием перечисленных ниже функциональных средств.

- Команда `complete`, позволяющая предоставить *спецификацию автозавершения* для отдельных команд. С помощью различных параметров можно указать, как приспособить список возможных вариантов завершения конкретной команды. Это делается очень просто и вполне пригодно для многих потребностей (Подробнее см. далее описание команды `complete` в разделе "Встроенные команды".)
- Для большего удобства можно воспользоваться командой `complete -F` *`имя_функции команда`*. Она предписывает оболочке Bash вызвать функцию, имеющую заданное *`имя_функции`*, чтобы предоставить список вариантов завершения указанной *`команды`*. Функцию, имеющую заданное *`имя_функции`*, следует написать самостоятельно.
- В коде функции, задаваемой с помощью параметра **`-F`**, переменные оболочки `COMP*` предоставляют сведения о текущей командной строке. В частности, переменная `COMPREPLY` представляет массив, в котором заданная функция размещает окончательный список результатов автозавершения.
- Кроме того, в коде функции, задаваемой с помощью параметра **`-F`**, можно воспользоваться командой `compgen`, чтобы сформировать список результатов автозавершения, например, список имен пользователей, начинающихся с буквы **`a`**, или всех установленных переменных.

Цель состоит в том, чтобы присвоить подобные результаты элементам массива, как показано ниже.

```
...
COMPREPLY=( $( compgen параметры аргументы ) )
...
```

Спецификации автозавершения могут быть связаны с полным путем к команде или (чаще всего) с именем команды без всяких добавлений (например, /usr/bin/man или просто man). Попытки автозавершения совершаются в приведенном ниже порядке в зависимости от конкретных параметров, указанных в команде complete.

1. Если попытка автозавершения совершается в пустой введенной строке, оболочка Bash применяет спецификацию автозавершения, заданную по команде complete -E. В противном случае она переходит к следующей стадии.
2. Оболочка Bash распознает сначала команду. Если указан путь к команде, оболочка Bash выясняет, существует ли спецификация автозавершения для полного пути к команде. В противном случае она устанавливает имя команды по последней составляющей заданного пути и производит поиск спецификации автозавершения для имени команды.
3. Если спецификация автозавершения существует, оболочка Bash использует ее. В противном случае оболочка Bash использует спецификацию автозавершения, задаваемую по умолчанию в команде complete -D. Если и такая спецификация отсутствует, то оболочка Bash обращается к встроенным стандартным вариантам завершения вводимых команд.
4. Оболочка Bash выполняет действие, указанное в спецификации автозавершения, чтобы сформировать список возможных совпадений. Из этого списка выбираются только те варианты, которые содержат завершаемое слово в виде префикса, а далее из них формируется список

возможных вариантов завершения. Если указаны параметры **-d** и **-f**, то оболочка Bash использует переменную FIGNORE, чтобы отсеять неприемлемые совпадения.

5. Оболочка Bash формирует имена файлов, указанные с помощью параметра **-G**. Для просеивания результатов используется переменная FIGNORE, но не переменная GLOBIGNORE.
6. Оболочка Bash обрабатывает строку аргументов, предоставляемую с помощью параметра **-W**. Эта строка разделяется на части с помощью символов, выбираемых из переменной $IFS. Из получаемого в итоге списка предоставляются возможные варианты для завершения. Такой способ нередко применяется для предоставления списка параметров, которые принимает команда.
7. Оболочка Bash выполняет функции и команды, указанные с помощью параметров **-F** и **-C**. Для тех и других оболочка Bash устанавливает переменные COMP_LINE и COMP_POINT (см. выше раздел “Встроенные в оболочку переменные”). А для функции оболочки устанавливаются также переменные COMP_WORDS и COMP_CWORD.

   Кроме того, для функций и команд позиционный параметр **$1** предоставляет имя команды, аргументы которой подлежат автозавершению, позиционный параметр **$2** — завершаемое слово, а позиционный параметр **$2** — слово, предшествующее завершаемому. Оболочка Bash *не* просеивает результаты выполнения команды или функции, придерживаясь следующей последовательности действий:

   а) сначала выполняются функции, заданные с помощью параметра **-F**. Такая функция должна установить в переменной массива COMPREPLY список возможных вариантов завершения, который затем извлекается оттуда оболочкой Bash;

   б) далее команды, заданные с помощью параметра **-C**, выполняются в среде, эквивалентной подстановке

команд. Такая команда должна вывести построчно список возможных вариантов завершения. Встроенные знаки новой строки должны быть экранированы знаками обратной косой черты.

8. Как только список возможных вариантов завершения будет сформирован, оболочка Bash просеет полученные результаты в соответствии с параметром **-X**. В качества аргумента для параметра **-X** служит шаблон, по которому из списка исключаются совпадающие файлы. Если же шаблон предваряется знаком **!**, то его назначение изменяется на обратное, а следовательно, по нему в списке должны оставаться только совпадающие файлы. Значение параметра оболочки `nocasematch` должно быть проигнорировано.

   Знак **&** заменяется в шаблоне текстом завершаемого слова. Чтобы использовать этот знак буквально, его необходимо экранировать следующим образом: **\&**.

9. И наконец, оболочка Bash дополняет завершаемое слово любыми префиксами или суффиксами, предоставляемыми с помощью параметра **-P** или **-S**.

10. Если никаких совпадений не обнаружено и при этом задан параметр **-o** `dirnames`, оболочка Bash попытается совершить автозавершение имен каталогов.

11. А если задан параметр **-o** `plusdirs`, оболочка Bash *добавит* результат автозавершения имен каталогов в сформированный ранее список.

12. Если предоставляется спецификация автозавершения, то оболочка Bash, как правило, не пытается использовать ни варианты автозавершения по умолчанию, ни варианты автозавершения имен файлов, выбираемые по умолчанию из библиотеки `readline`. Но если:

    а) спецификация автозавершения не дает никаких результатов и при этом задан параметр **-o** `bashdefault`, то

оболочка Bash попытается использовать свои варианты автозавершения по умолчанию;

б) ни спецификация автозавершения, ни варианты автозавершения, задаваемые в Bash оболочке по умолчанию с помощью параметра **-o** bashdefault, не дают никаких результатов и при этом задан параметр **-o** default, то оболочка Bash попытается использовать варианты автозавершения имен файлов, выбираемые из библиотеки readline.

Спецификация автозавершения может быть видоизменена по команде compopt. Если эта спецификация используется без имен команд в процессе автозавершения, то она оказывает влияние на данный процесс.

Если функция оболочки, применяемая в качестве обработчика автозавершения, возвращает код завершения **124**, оболочка Bash повторяет процесс автозавершения с самого начала. Это наиболее полезно для динамического построения ряда вариантов завершения в используемом по умолчанию обработчике автозавершения (по команде complete -D) вместо загрузки этих вариантов в большом количестве во время запуска оболочки. В конце раздела "Programmable Completion" (Автозавершение вводимых команд) на оперативной странице руководства *bash*(1) приведен характерный тому пример.

Совет

Иан Макдональд (Ian Macdonald) накопил огромное множество спецификаций автозавершения, нередко распространяемых в файле /etc/bash_completion. Если такой файл отсутствует в вашей системе, установите его с помощью доступного в ней диспетчера пакетов. В противном случае можете загрузить этот файл по адресу http://bash-completion.alioth.debian.org/. Его стоит просмотреть.

Примеры

Ограничиться файлами для компилятора C и C++, исходными файлами ассемблера и перемещаемыми объектными файлами можно по следующей команде:

```
complete -f -X '!*.[Ccos]' gcc cc
```

А ограничиться подстановками, описываемыми на оперативных страницах руководства, доступных по команде man, можно следующим образом:

```
# Простой пример автозавершения для оперативных страниц
# руководства. Более сложный пример приведен в файле
# bash_completion. Здесь предполагается синтаксис man [num]
# для команды man.

shopt -s extglob        # Активизировать расширенное
                        # сопоставление с шаблоном

# Определить функцию автозавершения
_man () {
# Локальные переменные
local dir mandir=/usr/share/man

# Очистить ответный список
COMPREPLY=( )

# Если задан номер раздела ...
if [[ ${COMP_WORDS[1]} = +([0-9]) ]]
then
    # предоставить раздел: man 3 foo
    # искать в указанном каталоге
    dir=$mandir/man${COMP_WORDS[COMP_CWORD-1]}
else
    # в отсутствие раздела перейти по умолчанию к командам
    # искать в каталогах команд
    dir=$mandir/'man[18]'
fi
COMPREPLY=( $(
  # сформировать исходный список файлов
  find $dir -type f |

    # Удалить начальные каталоги
    sed 's;..*/;;' |
```

```
        # Удалить конечные суффиксы
        sed 's/\.[0-9].*$//' |

            # Сохранить те файлы, которые совпадают
            # с заданным префиксом
            grep "^${COMP_WORDS[$COMP_CWORD]}" |

            # отсортировать окончательный список
            sort
  ) )
}
# Связать данную функцию с командой man
complete -F _man man
```

Управление заданиями

Управление заданиями позволяет переносить приоритетные задания в фоновый режим, а фоновые — в приоритетный режим или приостанавливать выполнение заданий. Оно поддерживается во всех современных Unix-подобных системах, включая Mac OS X, GNU/Linux и BSD, где функциональные средства управления заданиями активизируются автоматически. Многие команды управления заданиями принимают в качестве аргумента *идентификатор задания*, который может быть указан перечисленными ниже способами.

`%n`	Номер ***n*** задания
`%s`	Задание, командная строка которого начинается с указанной символьной строки ***s***
`%?s`	Задание, командная строка которого содержит указанную символьную строку ***s***
`%%`	Текущее задание
`%+`	Текущее задание, то же, что и `%%`
`%`	Текущее задание, то же, что и `%%`
`%-`	Предыдущее задание

Для управления заданиями в оболочке Bash предоставляются перечисленные ниже команды (более подробно они описываются далее, в разделе “Встроенные команды”).

bg. Перенести текущее задание в фоновый режим.

fg. Перенести текущее задание в приоритетный режим.

jobs. Перечислить активные задания.

kill. Прервать выполнение задания.

stty tostop. Остановить фоновые задания, если они попытаются направить выводимые результаты на эмулятор терминала. (Следует, однако, иметь в виду, что команда stty не является встроенной.)

suspend. Приостановить выполнение оболочки управления заданиями (например, оболочки, созданной по команде su).

wait. Ожидать завершения фоновых заданий.

Комбинация клавиш <Ctrl+Z>. Приостановить выполнение приоритетного задания. Затем воспользоваться командой bg или fg. (Для приостановки выполнения приоритетного задания в вашем эмуляторе терминала может использоваться другая комбинация клавиш, хотя это маловероятно.)

Параметры оболочки

В оболочке Bash предоставляется целый ряд параметров, устанавливая которые можно изменить режим работы оболочки. Эти параметры устанавливаются с помощью команды shopt, более подробно описываемой далее, в разделе "Встроенные команды".

Параметры compat*NN* являются взаимоисключающими. Уровень совместимости обозначает минимальный уровень. Например, при установке параметра compat40 оболочка действует по версии Bash 4.0 в отношении тех функциональные средств, на которые оказывают влияние установки параметров совместимости, изменившиеся после версии 4.0. Поэтому предпочтительнее пользоваться переменной BASH_COMPAT.

Ниже приведено краткое описание режима работы оболочки при установке различных ее параметров. Те параметры, которые помечены крестиком (†), активизируются по умолчанию.

autocd. Если первое слово простой команды нельзя выполнить, следует попытаться перейти к нему, вызвав функцию cd. Если такая функция отсутствует, оболочка выполнит вместо нее встроенную команду cd.

cdable_vars. Интерпретировать некаталожный аргумент команды cd как переменную, значением которой является каталог для перехода.

cdspell. Попытаться исправить орфографические ошибки в написании каждой составляющей пути, указываемого в качестве аргумента команды cd. Допускается только в интерактивных оболочках.

checkhash. Проверять, существуют ли еще команды в хеш-таблице, прежде чем пытаться их использовать. Если они отсутствуют, выполнить обычный поиск по содержимому переменной окружения $PATH.

checkjobs. Если предпринимается попытка выйти из оболочки и при этом существуют приостановленные или выполняющиеся фоновые задания, оболочка выведет сообщение "There are running jobs" (Имеются выполняющиеся задания) и перечислит как сами задания, так и их состояния. При повторной попытке выйти из оболочки (например, при повторном вводе признака конца файла) произойдет выход из оболочки.

checkwinsize. Проверять размер окна после каждой команды и обновить содержимое переменных LINES и COLUMNS, если размер окна изменился. Действует как в интерактивных, так и в неинтерактивных оболочках.

cmdhist †. Сохранять все строки многострочной команды в одной записи предыстории. Это позволяет легко редактировать многострочные команды.

compat31. Восстанавливать режим работы операции присваивания **=~** для команды `[[ ]]`. Благодаря этому правый операнд всегда интерпретируется как регулярное выражение, с которым производится сопоставление. Кроме того, в операциях **<** и **>** сравнения символьных строк игнорируются региональные настройки на языковой стандарт.

compat32. Игнорировать региональные настройки на языковой стандарт для команды `[[ ]]` в операциях **<** и **>** сравнения символьных строк. Кроме того, прерывание выполнения команды посредине списка команд (например, `cmd1; cmd2; cmd3`) не приводит к прерыванию выполнения всего списка команд.

compat40. Игнорировать региональные настройки на языковой стандарт для команды `[[ ]]` в операциях **<** и **>** сравнения символьных строк.

compat41. В режиме работы по стандарту POSIX интерпретировать одиночные кавычки, имеющиеся в подстановке параметров в двойных кавычках, как символы заключения в кавычки. Количество одиночных кавычек должно быть четным, а их содержимое интерпретируется как заключаемое в кавычки.

compat42. Не обрабатывать замещающую строку в подстановке слов по шаблонам, удаляя кавычки.

compat43. Не выводить предупреждающее сообщение, если в качестве аргумента команды `declare` указывается операция составного присваивания в кавычках. Интерпретировать ошибки подстановки слов как исправимые и приводящие к неудачному завершению команды — даже в режиме работы по стандарту POSIX. Кроме того, не устанавливать в исходное состояние цикл в теле функции, чтобы команды `break` и `continue` в теле функции оказывали влияние на выполнение циклов в коде, вызывающем данную функцию.

complete_fullquote †. Экранировать все употребляемые в оболочке метасимволы подстановки знаками обратной косой черты при автозавершении имен файлов. Если этот параметр дезактивизирован, знаки денежной единицы, а возможно, и другие знаки, не экранируются, и поэтому подстановка переменных оболочки производится как обычно.

direxpand. Заменить имена каталогов результатами подстановки слов при автозавершении имен файлов, видоизменив буфер редактирования из библиотеки `readline`.

dirspell. Попытаться исправить орфографические ошибки в написании имен каталогов при автозавершении слов, если имя в заданном виде не существует.

dotglob. Включать имена файлов, начинающиеся с точки, в результаты подстановки имен файлов.

execfail. Не выходить из неинтерактивной оболочки, если нельзя выполнить команду, заданную в качестве аргумента команды `exec`. Выход из интерактивных оболочек в данном случае не происходит независимо от установки этого параметра.

expand_aliases †. Подставить псевдонимы, созданные по команде `alias`. Этот параметр дезактивизирован в неинтерактивных оболочках.

extdebug. Активизировать режим работы, требующийся для отладчиков. В частности:

Команда `declare -F` отображает имя исходного файла и номер строки для каждого аргумента функции с заданным именем.

- Если выполнение команды по прерыванию DEBUG завершается неудачно, то следующая команда пропускается.
- Если команда, выполняемая по прерыванию DEBUG в теле функций оболочки или сценариев, вызываемых

по встроенной команде . (точка) или source, возвращает код завершения 2, оболочка имитирует обращение к команде return.

- Переменные BASH_ARGC и BASH_ARGV устанавливаются, как описано ранее.
- Активизируется трассировка функций. Подстановки команд, функции оболочки и подоболочки, вызываемые в форме (...), наследуют прерывания DEBUG и RETURN.
- Активизируется трассировка ошибок. Подстановки команд, функции оболочки и подоболочки, вызываемые в форме (...), наследуют прерывание ERR.

extglob. Активизировать расширенные средства сопоставления с шаблоном, например, форму +(...). Этот параметр автоматически активизируется в режиме работы по стандарту POSIX. (Эти средства отсутствовали в первоначальной оболочке Bourne shell, и поэтому оболочка Bash требует их активизации вручную, если они действительно нужны.)

extquote †. Разрешить применение форм $'...' и $"..." в подстановках ${*переменная*}, заключаемых в двойные кавычки.

failglob. Обусловить выдачу ошибок шаблонами, не совпадающими с именами файлов.

force_fignore †. Игнорировать при автозавершении слова, совпадающие со списком суффиксов в переменной FIGNORE, даже если они являются единственно возможными вариантами завершения.

globasciiranges. Расширить пределы в выражениях, заключаемых в квадратные скобки для сопоставления с шаблоном, как будто они относятся к региональным настройкам "C", проигнорировав последовательности сортировки из текущих региональных настроек. Этот

параметр будет активизирован по умолчанию в версии Bash 5.0.

globstar. Активизировать расширенное сопоставление с каталогами и подкаталогами по специальному шаблону **.

gnu_errfmt. Выводить сообщения об ошибках в стандартном для общей лицензии GNU формате. Этот параметр автоматически активизируется при выполнении оболочки Bash в окне терминала текстового редактора Emacs.

histappend. Присоединять список предыстории выполнения команд к файлу, имя которого определяется переменной `$HISTFILE`, при выходе из оболочки, вместо того, чтобы перезаписывать этот файл.

histreedit. Разрешить пользователю повторно редактировать неудачную подстановку предыстории в стиле оболочки `csh` средствами библиотек `readline`.

histverify. Выводить результаты подстановки предыстории в стиле оболочки `csh` в буфер редактирования из библиотеки `readline` вместо того, чтобы выполнять эту подстановку непосредственно, на тот случай, если пользователь пожелает внести дополнительные изменения.

hostcomplete †. Если применяется библиотека `readline`, то попытаться совершить автозавершение имени хоста при автозавершении слова, содержащего знак `@`.

huponexit. Посылать сигнал `SIGHUP` всем выполняющимся заданиям при выходе из интерактивной исходной оболочки.

inherit_errexit. Обусловить наследование значения параметра `errexit`, устанавливаемого по команде `set -e`, при подстановке команд. Этот параметр автоматически активизируется в режиме работы по стандарту POSIX.

interactive_comments †. Разрешить интерпретировать слова, начинающиеся со знака **#**, как последующие комментарии в интерактивной оболочке.

lastpipe. Выполнять в оболочках без управления заданиями последнюю команду из приоритетного конвейера в рабочей среде текущей оболочки. Все команды, кроме последней, выполняются в подоболочках.

lithist. Если параметр `cmdhist` не установлен, сохранять многострочные команды в файле предыстории, заменяя точки с запятой знаками новой строки.

login_shell. Этот параметр устанавливается оболочкой в том случае, если она запускается как исходная. Его значение доступно только для чтения.

mailwarn. Выводить сообщение "`The mail in` *`mailfile`* `has been read`" (Почта из почтового файла прочитана), когда файл, проверяемый на наличие почты, доступен с момента последней проверки почты в оболочке Bash.

no_empty_cmd_completion. Если применяется библиотека `readline`, *не* производить поиск по содержимому переменной окружения `$PATH` при попытке автозавершения пустой строки или же строки, состоящей только из пробелов.

nocaseglob. Игнорировать регистр букв при сопоставлении имен файлов.

nocasematch. Игнорировать регистр букв при сопоставлении по шаблону в командах `case` и `[[ ]]`.

nullglob. Подставлять вместо шаблонов, не совпадающих ни с одним из файлов, нулевую строку вместо того, чтобы использовать сами шаблоны в качестве аргументов.

progcomp †. Активизировать автозавершение.

promptvars †. Выполнять подстановку переменных, команд и арифметических выражений в строковых значениях переменных `PS0`, `PS1`, `PS2` и `PS4`.

restricted_shell. Этот параметр устанавливается оболочкой, когда она оказывается ограниченной. Его значение доступно только для чтения.

shift_verbose. Обусловить вывод из команды `shift` сообщения об ошибке, когда подсчитанное количество сдвигов превышает количество позиционных параметров.

sourcepath †. Обусловить поиск в командах `.` (точка) и `source` по содержимому переменной окружения `$PATH`, чтобы найти файл, доступный для чтения и выполнения.

xpg_echo. Обусловить подстановку в команде `echo` управляющих последовательностей, даже если в ней не задан параметр **-e** или **-E**.

Выполнение команд

Когда вводится команда, оболочка Bash ищет ее в следующих указанных по порядку местах до тех пор, пока не обнаружит совпадение.

1. Ключевые слова вроде `if` и `for`.
2. Псевдонимы. В режиме работы по стандарту POSIX нельзя определить псевдоним, имя которого совпадает с ключевым словом оболочки, но в то же время можно определить псевдоним, подставляющий ключевое слово (например, `aslongas=while`). А в режиме работы не по стандарту POSIX оболочка позволяет определить псевдоним для ключевого слова `for`.

   Как правило, подстановка псевдонимов разрешается только в интерактивных оболочках. А в оболочках по стандарту POSIX она разрешена всегда.
3. Специальные встроенные команды вроде `break` и `continue` (только в оболочках по стандарту POSIX). К их числу относятся команды `.` (точка), `:` (двоеточие), `break`,

continue, eval, exec, exit, export, readonly, return, set, shift, times, trap и unset. В оболочке Bash их число дополнено командой source. Если выполнение специальной команды, встроенной в оболочку по стандарту POSIX, приводит к ошибке, то происходит немедленный выход из неинтерактивных оболочек.

4. Функции. В режиме работы не по стандарту POSIX оболочка Bash обнаруживает функции прежде *всех* встроенных команд.
5. Неспециальные встроенные команды вроде cd и test.
6. Сценарии и исполняемые программы, для которых оболочка производит поиск команды в каталогах, перечисленных в переменной окружения $PATH. *Примечание.* В режиме работы по стандарту POSIX знаки тильды в элементах пути, указанных в переменной окружения $PATH, не подлежат подстановке. А если команда уже отсутствует в хеш-таблице, то оболочка производит повторный поиск по содержимому переменной окружения $PATH.
7. Если команда не найдена и при этом существует функция command_not_found_handle, то оболочка вызывает ее, передавая ей слова искомой команды в качестве аргументов.

Отличие специальных встроенных команд от неспециальных берет свое начало из стандарта POSIX. Это отличие в сочетании с командой command дает возможность создавать функции, переопределяющие встроенные в оболочку команды вроде cd. Например:

```
# Функция оболочки; обнаруживается прежде
# встроенной команды cd
cd () {
    command cd "$@"      # Воспользоваться настоящей
                         # командой cd,
                         # чтобы сменить каталог
    echo now in $PWD     # Другая команда, которую
                         # требуется выполнить
}
```

Если выход из оболочки Bash происходит вследствие поступления сигнала SIGHUP или же если установлен параметр оболочки huponexit, то оболочка Bash посылает сигнал SIGHUP всем выполняющимся порожденным заданиям. Чтобы воспрепятствовать оболочке Bash посылать сигнал SIGHUP конкретному заданию, достаточно выполнить команду disown -h.

Сопроцессы

Сопроцессом называется такой процесс, который выполняется параллельно с оболочкой и с которым она взаимодействует. Оболочка запускает такой процесс в фоновом режиме, соединяя его стандартный ввод и вывод в двунаправленный канал. (Стандартный вывод ошибок из сопроцесса не переадресовывается.)

Для выполнения сопроцессов имеются следующие синтаксические формы:

```
coproc имя непростая команда      # Запустить именованный
                                  # процесс
coproc команда аргументы          # Запустить неименованный
                                  # процесс
```

Оболочка создает переменную массива, получающую заданное *имя* и предназначенную для хранения дескрипторов взаимодействия с сопроцессом. В частности, элемент массива *имя*[0] содержит вывод из сопроцесса (и ввод в контролирующую оболочку), а элемент массива *имя*[1] — ввод в сопроцесс (и вывод из оболочки). Кроме того, в переменной *имя*_PID хранится идентификатор сопроцесса. Если же *имя* не указано, то оболочка использует переменную массива COPROC.

ПРИМЕЧАНИЕ

Одновременно может действовать только один сопроцесс.

Пример

В следующем примере демонстрируется элементарное применение ключевого слова coproc и связанных с ним переменных:

```
# Запустить именованный сопроцесс в фоновом режиме
$ coproc testproc (echo 1
> read aline ; echo $aline)
[1] 5090

# Показать дескрипторы файлов
$ echo ${testproc[@]}
63 60

# Показать идентификатор сопроцесса
$ echo $testproc_PID
5090

# Прочитать первую строку, выводимую из сопроцесса,
# и показать ее
$ read out <&${testproc[0]}
$ echo $out
1
# Прочитать некоторые данные, вводимые в сопроцесс
$ echo foo >&${testproc[1]}

# Прочитать вторую строку, выводимую из сопроцесса
$ read out2 <&${testproc[0]}
[1]+ Done coproc testproc (echo 1; read aline; echo
$aline)

# Показать вторую строку, выводимую из сопроцесса
$ echo $out2
foo
```

Ограниченные оболочки

Ограниченной называется такая оболочка, где запрещаются определенные действия, например, смена каталога, установка

переменной окружения `$PATH` или выполнение команд, в именах которых содержится знак **/**.

В первоначальной версии оболочки V7 Bourne shell имелся недокументированный ограниченный режим работы. А в последующих ее версиях такой режим был прояснен и задокументирован. В оболочке Bash также поддерживается ограниченный режим работы (подробнее об этом см. соответствующую оперативную страницу руководства).

В ограниченном режиме работы по-прежнему можно выполнять сценарии оболочки, поскольку в этом случае из ограниченной оболочки вызывается ее неограниченная версия для выполнения сценария. С этой целью используются файлы `/etc/profile`, `~/.profile` и прочие файлы запуска.

Совет

На практике ограниченные оболочки применяются нечасто, так как правильно настроить их не так-то просто.

Встроенные команды

В приведенных далее примерах встроенных команд их ввод демонстрируется с приглашением **$**. В противном случае эти примеры следует рассматривать как фрагменты кода, которые могут быть включены в сценарий оболочки. Ради удобства указываются также зарезервированные слова, используемые в многострочных командах. Почти все встроенные команды распознают параметр **`--help`** и выводят в ответ сводку об их применении.

! **Изменить на обратное назначение последующего конвейера**

`!` *`конвейер`*

Выполнить отрицание назначения указанного конвейера. Возвращает нулевой код завершения, если конвейер

завершается ненулевым кодом, или единичный код завершения, если конвейер завершается нулевым кодом. Как правило, эта команда применяется в условном операторе `if` и в операторе цикла `while`.

Пример

В следующем фрагменте кода выводится сообщение, если пользователь `jane` не вошел в систему:

```
if ! who | grep jane > /dev/null
then
    echo jane is not currently logged on
fi
```

— **Вставить комментарий до конца строки**

`#` *`текст`* `...`

Игнорировать весь текст до конца данной строки. Команда **#** применяется в сценариях оболочки в качестве символа комментария и на самом деле не является командой.

#! ***оболочка*** — **Вызвать именованный интерпретатор для выполнения оболочки**

`#!`*`оболочка`* `[`*`параметр`*`]`

Служит в качестве первой строки сценария для вызова указанной *оболочки*. Все, что задано в остальной части строки, передается указанной *оболочке* в качестве *единственного аргумента*. Например:

```
#!/bin/sh
```

Совет

Эта команда, как правило, реализуется на уровне ядра, но может и не поддерживаться в ряде очень старых систем. А в некоторых системах накладывается ограничение около **32** символов на максимальную длину имени указываемой *оболочки*.

: **Пустая команда, применяемая в качестве синтаксического заполнителя**

```
: [аргументы]
```

Это пустая команда, возвращающая нулевой код завершения. Ее применение демонстрируется в приведенном ниже примере и далее при описании команды `case`. Строка с данной командой по-прежнему обрабатывается с побочными эффектами вроде подстановок переменных и команды или переадресации ввода-вывода.

Пример

Проверить, вошел ли кто-нибудь в систему:

```
if who | grep $1 > /dev/null
then :   # Ничего не делать, если в системе обнаружен
         # пользователь
else echo "User $1 is not logged in"
fi
```

. **Прочитать и выполнить файл в текущей оболочке**

```
. файл [аргументы]
```

Прочитать и выполнить строки из указанного *файла*, которому совсем не обязательно быть исполняемым, тем не менее, он должен присутствовать в каталоге, искомом по содержимому переменной окружения `$PATH`. Если параметр `sourcepath` дезактивизирован, оболочка Bash не производит поиск по содержимому переменной окружения `$PATH`. Заданные *аргументы* сохраняются в позиционных параметрах. Если указанный *файл* отсутствует в переменной окружения `$PATH`, оболочка Bash ищет его в текущем каталоге. (Следует, однако, иметь в виду, что оболочки по стандарту POSIX этого не делают.) Оболочка Bash удалит нулевые байты (NUL в коде ASCII) из содержимого указанного *файла*, прежде чем пытаться произвести его синтаксический анализ. Если указанный *файл* не найден, происходит выход из неинтерактивных оболочек, работающих в режиме по стандарту POSIX, если только он не

предварен командой `command`. См. также приведенное далее описание команды `source`.

Если установлен параметр **`-T`**, то наследуются прерывания по сигналу `DEBUG`, и любые изменения, которые указанный *файл* вносит в прерывание `DEBUG`, остаются в силе вплоть до возврата в вызывающую оболочку. Если же параметр **`-T`** не установлен, то прерывания `DEBUG` сохраняются и восстанавливаются при обращении к указанному *файлу*, а само прерывание `DEBUG` сбрасывается во время выполнения указанного *файла*.

`[[ ]]` **Расширенная версия команды `test`**

`[[` *`выражение`* `]]`

То же, что и `test` *`выражение`* или `[` *`выражение`* `]`, за исключением того, что в команде `[[ ]]` допускается указывать дополнительные операции. Разделение слов и подстановка имен файлов запрещается. Но следует иметь в виду, что внешние квадратные скобки (**`[ ]`**) вводятся буквально и должны быть отделены пробелами. Подробнее об этом см. далее описание команды `test`.

Дополнительные операции

- **`&&`** Укороченная логическая операция И для проверки выражений
- **`||`** Укороченная логическая операция ИЛИ для проверки выражений
- **`<`** Операция сравнения, где первая символьная строка должна быть лексически *меньше* второй строки в зависимости от порядка сортировки, установленного в региональных настройках. (Впрочем, см. описание параметров **`compat31`**, **`compat32`** и **`compat40`**, приведенное ранее в разделе "Параметры оболочки".)
- **`>`** Операция сравнения, где первая символьная строка должна быть лексически *больше* второй строки в зависимости от порядка сортировки, установленного в региональных настройках. (Впрочем, см. описание параметров **`compat31`**, **`compat32`** и **`compat40`**, приведенное ранее в разделе "Параметры оболочки".)

имя **`()`** **Определить функцию оболочки**

`имя` `() {` *`команды;`* `} [`*`переадресации`*`]`

Определить указанное *имя* как функцию, используя синтаксис по стандарту POSIX. Определение функции может быть как однострочным, так и многострочным. Можно также

указать ключевое слово `function` в качестве альтернативной формы определения функции, действующей аналогичным образом. Подробнее об этом. см. выше в разделе “Функции”.

Пример

После ввода приведенного ниже определения функции `countfiles` в командной строке она отобразит количество файлов, находящихся в текущем каталоге.

```
$ countfiles () {
>     ls | wc -l
> }
```

имя_файла **Выполнить внешнюю команду**

имя_файла [*аргументы*]

Прочитать и выполнить команды из внешнего файла, имеющее указанное *имя_файла*, или же выполнить аналогично указанный двоичный объектный файл. Если указанное *имя_файла* не содержит знаки косой черты, оболочка ищет файл для выполнения в каталогах, перечисленных в переменной окружения `$PATH`.

`alias` **Определить псевдонимы и управлять ими в оболочке**

`alias [-p] [`*имя*`[='`*команда*`']]`

Присвоить заданной *команде* сокращенное *имя* в качестве ее синонима. Если выражение `='`*команда*`'` опущено, вывести псевдоним указанного *имени*, а если и оно опущено, то вывести все псевдонимы. Если значение псевдонима содержит конечный пробел, то следующее слово в командной строке также подлежит подстановке псевдонима. Переменная массива `BASH_ALIASES` обеспечивает программный доступ ко всем определенным псевдонимам (подробнее об этом см. выше в разделе “Встроенные в оболочку переменные”). См. также приведенное далее описание команды `unalias`.

Совет

В общем, функции предпочтительнее псевдонимов, поскольку позволяют использовать локальные переменные в своем теле и полностью поддаются программированию.

Параметр

-p Вывести слово **alias** перед каждым псевдонимом

Пример

```
alias dir='echo ${PWD##*/}'
```

---

bg **Перенести приостановленное задание в фоновый режим**

bg [*идентификаторы заданий*]

Перенести текущее задание или несколько заданий, имеющих указанные *идентификаторы заданий*, на выполнение в фоновый режим. Подробнее об этом см. выше в разделе “Управление заданиями”.

---

bind **Управлять привязками оперативных клавиш из библиотеки** readline

```
bind [-m раскладка] [параметры]
bind [-m раскладка] [-q функция] [-r последовательность]
                    [-u функция]
bind [-m раскладка] -f файл
bind [-m раскладка] -x последовательность:команда
bind [-m map] последовательность:функция
bind команда из библиотеки readline
```

Управлять привязками оперативных клавиш и функций из библиотеки readline. Аргументы, не относящиеся к параметрам, имеют такую же форму, как и в файле .inputrc.

Параметры

-f *файл*
Читать привязки оперативных клавиш из указанного ***файла***

-l Перечислить имена всех функций из библиотеки **readline**

-m *раскладка*

Использовать заданную ***раскладку*** в качестве раскладки оперативных клавиш. Доступными являются раскладки оперативных клавиш **emacs**, **emacs-ctlx**, **emacs-standard**, **emacs-meta**, **vi**, **vi-command**, **vi-insert** и **vi-move**, где **vi** — то же самое, что и **vi-command**, а **emacs** — то же самое, что и **emacs-standard**

-p Вывести текущие привязки из библиотеки **readline** таким образом, чтобы их можно было прочитать повторно из файла **.inputrc**

-P Вывести текущие привязки из библиотеки **readline**

-q *функция*

Запросить оперативные клавиши, предназначенные для вызова указанной ***функции*** из библиотеки **readline**

-r *последовательность*

Удалить привязку к заданной ***последовательности*** оперативных клавиш

-s Вывести текущую последовательность оперативных клавиш и макропривязки из библиотеки **readline** таким образом, чтобы их можно было прочитать повторно из файла **.inputrc**

-S Вывести текущую последовательность оперативных клавиш и макропривязки из библиотеки **readline**

-u *функция*

Отвязать все оперативные клавиши, предназначенные для вызова указанной ***функции*** из библиотеки **readline**

-v Вывести текущие переменные из библиотеки **readline** таким образом, чтобы их можно было прочитать повторно из файла **.inputrc**

-V Вывести текущие переменные из библиотеки **readline**

-x *последовательность*:*команда*

Выполнять заданную ***команду*** оболочки всякий раз, когда вводится указанная ***последовательность***. В заданной ***команде*** можно применять и видоизменять переменные **READLINE_LINE** и **READLINE_POINT**. Изменения в этих переменных отражаются на состоянии редактирования

-X Вывести текущие последовательности оперативных клавиш, привязанные из библиотеки **readline** с помощью параметра **-x**, таким образом, чтобы их можно было прочитать повторно из файла **.inputrc**

---

break — Выйти из одного или нескольких циклов

`break [n]`

Выйти из цикла `for`, `while`, `select` или `until` (или прервать *n* вложенных циклов).

builtin **Выполнить встроенную команду в обход функций**

```
builtin команда [аргументы ...]
```

Выполнить указанную *команду*, встроенную в оболочку, вместе с заданными *аргументами*. Это дает возможность обойти любые функции, переопределяющие имя встроенной команды. Указанная *команда* является более переносимой, чем соответствующие функции.

Пример

Следующая функция позволяет выполнить нужные задачи при смене каталога:

```
cd () {
    builtin cd "$@"        # Фактически сменить каталог
    pwd                    # Сообщить новое местоположение
}
```

caller **Вывести функцию или код, вызывающий файл с точкой, для применения в отладчике оболочки Bash**

```
caller [выражение]
```

Вывести номер строки вместе с именем исходного файла, содержащего текущий вызов функции или файла с точкой. Если указано ненулевое *выражение*, то выводится соответствующий элемент из стека вызовов. Самым последним в стеке вызовов является нулевой элемент. Эта команда предназначена для применения в отладчике оболочки Bash.

case **Синтаксис для одноименного оператора выбора**

```
case значение in
[(]шаблон1) команды1;;          # ;& или ;;& -- см. описание
[(]шаблон2) команды2;;
. . .
esac
```

Выполнить первый ряд команд (*команды*$_1$), если указанное *значение* совпадает с одним заданным шаблоном (*шаблон*$_1$); второй ряд команд (*команды*$_2$), если указанное *значение* совпадает с другим заданным шаблоном (*шаблон*$_2$), и т.д. Каждый

ряд команд должен оканчиваться знаками `;;`. Как правило, указывается *значение* позиционного параметра или другой переменной оболочки, а *команды* являются исполняемыми командами, командами языка программирования оболочки или операциями присваивания значений переменным. В задаваемых шаблонах можно употреблять метасимволы подстановки имен файлов. В одной строке можно указать несколько шаблонов, разделив их знаками `|`. В таком случае соответствующие *команды* выполняются всякий раз, когда указанное *значение* совпадает с любым из этих шаблонов. См. примеры, приведенные ниже и далее в описании команды `eval`.

В оболочке Bash допускается предварять заданный *шаблон* дополнительной, хотя и не обязательной открывающей скобкой, как, например, `(`*шаблон*`)`. В некоторых версиях данной оболочки необходимо соблюдать парность скобок в конструкции `$( )`, но, начиная с версии Bash 4.0, этого больше не требуется. См. также описание параметра `nocasematch` ранее в разделе "Параметры оболочки".

В оболочке Bash предоставляются два дополнительных варианта окончания задаваемого ряда *команд* в операторе выбора `case`. В частности, знаки `;&` обозначают необходимость продолжить выполнение со следующего ряда *команд*, а знаки `;;&` — продолжить проверку по следующему списку *шаблонов*.

Примеры

Проверить первый аргумент командной строки и предпринять соответствующее действие:

```
case $1 in      # Проверить на совпадение первый аргумент
no|yes)   response=1;;
-[tT])    table=TRUE;;
*)        echo "unknown option"; exit 1;;
esac
```

Читать введенные пользователем строки до тех пор, пока не завершится их ввод:

```
while true
do printf "Type . to finish ==> "
```

```
    read line
    case "$line" in
    .)   echo "Message done"
         break ;;
    *)   echo "$line" >> $message ;;
    esac
done
```

cd **Сменить каталог**

```
cd [-L] [-P [-e]] [-@] [каталог]
cd [-L] [-P [-e]] [-@] [-]
```

В отсутствие аргументов перейти в начальный каталог текущего пользователя. В противном случае сменить рабочий каталог на указанный *каталог*. Оболочка Bash производит поиск указанного *каталога* сначала по содержимому переменной окружения `$CDPATH`, а затем в текущем каталоге. Если *каталог* указан по относительному пути, но отсутствует в текущем каталоге, то его поиск также производится по содержимому переменной окружения `$CDPATH`. Имя каталога - обозначает предыдущий каталог. Если содержимое переменной `PWD` доступно только для чтения, то данная команда завершается с кодом неудачного завершения.

Параметры

-e Если задан параметр **-P**, а текущий каталог нельзя определить, завершить команду с кодом неудачного завершения

-L Использовать логический путь (т.е. то, что введено пользователем, включая любые символические ссылки) в команде **cd ..** и значение переменной **PWD**. Это делается по умолчанию

-P Использовать физический путь из файловой системы в команде **cd ..** и значение переменной **PWD**

-@ Интерпретировать файл с расширенными атрибутами в тех системах, где они поддерживаются, как каталог, содержащий атрибуты данного файла

Примеры

```
$ ls -ld /var/run    # /var/run - это символическая ссылка
lrwxrwxrwx 1 root root 4 May 7 19:41 /var/run -> /run
$ cd -L /var/run     # Логическая смена каталога
```

```
$ pwd                  # Показать текущее местоположение
/var/run               # Результат — логическое местоположение
$ cd -P /var/run       # Физическая смена каталога
$ pwd                  # Показать текущее местоположение
/run                   # Результат — физическое местоположение
```

command **Выполнить встроенную команду или вывести сведения о ней**

command [-pvV] *имя* [*аргументы* ...]

Если не указан параметр **-v** или **-V**, выполнить команду, имеющую указанное *имя*, с заданным *аргументами*. Данная команда действует в обход любых псевдонимов или функций, которые могут быть определены для выполняемой команды, имеющей указанное *имя*. Если данная команда применяется для выполнения специальной встроенной команды, она предотвращает выход из сценария при неудачном завершении встроенной команды. В режиме работы по стандарту POSIX операции присваивания, задаваемые в качестве аргументов в командах alias, declare, export, local, readonly и typeset, по-прежнему возымеют действие, даже если им предшествует команда command.

Параметры

-p Использовать предопределенный по умолчанию путь поиска, а не текущее значение переменной окружения **$PATH**

-v Вывести описание, поясняющее, каким образом команда, имеющая указанное ***имя***, интерпретируется в оболочке

-V Вывести более подробное описание, поясняющее, каким образом команда, имеющая указанное ***имя***, интерпретируется в оболочке

Пример

Создать псевдоним команды rm, получающей версию системы, и выполнить ее с параметром **-i**:

```
$ alias 'rm=command -p rm -i'
```

compgen **Сформировать возможные варианты автозавершения**

compgen [*параметры*] [*строка*]

Сформировать возможные варианты автозавершения указанной *строки* в соответствии с заданными *параметрами*. Принимаются все параметры команды `complete`, кроме параметров **`-p`** и **`-r`**. Подробнее об этом см. приведенное ниже описание команды `complete`.

`complete` **Указать, каким образом совершается автозавершение ввода отдельных команд**

`complete [-DE] [`*`параметры`*`]` *`команда`* ...

Указать, каким образом следует завершить ввод аргументов каждой заданной *команды*. Подробнее об автозавершении см. выше в разделе "Автозавершение вводимых команд".

Параметры

`-a` То же, что и **`-A alias`**

`-A` ***`тип`***

Использовать указанный ***`тип`*** для обозначения списка возможных вариантов автозавершения. Указанный ***`тип`*** может быть одним из следующих:

`alias`	Имена псевдонимов
`arrayvar`	Имена переменных массивов
`binding`	Привязки из библиотеки **`readline`**
`builtin`	Имена команд, встроенных в оболочку
`command`	Имена команд
`directory`	Имена каталогов
`disabled`	Имена запрещенных команд, встроенных в оболочку
`enabled`	Имена разрешенных команд, встроенных в оболочку
`export`	Экспортируемые переменные
`file`	Имена файлов
`function`	Имена функций оболочки
`group`	Имена групп
`helptopic`	Разделы справки, допустимые по встроенной команде **`help`**
`hostname`	Имена хостов (т.е. сетевых узлов), находящиеся в файле, имя которого определяется переменной **`$HOSTFILE`**
`job`	Имена заданий
`keyword`	Ключевые слова, зарезервированные в оболочке
`running`	Имена выполняющихся заданий

service	Имена служб, доступных по пути **/etc/services**
setopt	Достоверные аргументы команды **set -o**
shopt	Имена достоверных параметров встроенной команды **shopt**
signal	Имена сигналов
stopped	Имена приостановленных заданий
user	Имена пользователей
variable	Имена переменных оболочки

-b То же, что и **-A builtin**

-c То же, что и **-A command**

-C ***команда***

Выполнить указанную ***команду*** в подоболочке и воспользоваться выводимым из нее результатом в качестве списка возможных вариантов автозавершения

-d То же, что и **-A directory**

-D Применить остальные параметры для "стандартного" автозавершения, которое совершается по умолчанию, если не обнаружено других спецификаций автозавершения

-e То же, что и **-A export**

-E Применить остальные параметры для "пустого" автозавершения, которое совершается при попытке автоматически завершить пустую введенную строку

-f То же, что и **-A file**

-F ***функция***

Выполнить указанную ***функцию*** в текущей оболочке. После возврата из этой функции извлечь список вариантов автозавершения из переменной массива **COMPREPLY**

-g То же, что и **-A group**

-G ***шаблон***

Подставить заданный ***шаблон***, чтобы сформировать варианты автозавершения

-j То же, что и **-A job**

-k То же, что и **-A keyword**

-o ***параметр***

Определить поведение спецификации автозавершения. Указанный ***параметр*** может принимать одно из следующих значений:

bashdefault	Вернуться к обычным для оболочки Bash вариантам автозавершения, если ничего другого не обнаружено
default	Использовать стандартные для библиотеки **readline** варианты автозавершения, если ничего другого не обнаружено
dirnames	Совершить автозавершение имен каталогов, если ничего другого не обнаружено

`filenames`	Известить библиотеку **`readline`** о намерении вывести имена файлов для последующей их обработки средствами данной библиотеки, включая добавление конечного знака косой черты для обозначения каталогов или удаление конечных пробелов
`noquote`	Известить библиотеку **`readline`** не заключать в кавычки завершенные слова, обозначающие имена файлов
`nosort`	Известить библиотеку **`readline`** не сортировать список завершенных слов
`nospace`	Известить библиотеку **`readline`** не дополнять завершенные слова пробелами до конца строки
`plusdirs`	Попытаться автоматически завершить имя каталога и ввести любые результаты в список уже сформированных вариантов автозавершения

`-p` В отсутствие команд вывести все настройки автозавершения таким образом, чтобы их можно было прочитать повторно

`-P` ***префикс***

Предварить заданным ***префиксом*** каждую символьную строку, получающуюся после применения всех остальных параметров

`-r` Удалить настройки автозавершения заданных команд или же все настройки в отсутствие команд

`-s` То же, что и **`-A service`**

`-S` ***суффикс***

Присоединить указанный ***суффикс*** к каждой символьной строке, получающейся после применения всех остальных параметров

`-u` То же, что и **`-A user`**

`-v` То же, что и **`-A variable`**

`-W` ***список_слов***

Разделить заданный (одним словом в оболочке) ***список_слов***, используя символ из переменной **`$IFS`**. Сформированный в итоге список будет содержать отдельные элементы разделенного списка, совпадающие с завершаемым словом. Каждый элемент разделяемого списка затем подставляется посредством раскрытия скобок, замены знака тильды, подстановки параметров, переменных, команд и арифметических выражений. При этом во внимание принимается заключение в кавычки

`-X` ***шаблон***

Исключить имена файлов, совпадающие с заданным ***шаблоном***, из списка вариантов автозавершения имен файлов. Если заданный ***шаблон*** предваряется знаком **`!`**, его назначение меняется на обратное, и тогда в списке сохраняются только те имена файлов, которые совпадают с заданным ***шаблоном***

Совет

С любезного разрешения Чета Рэйми (Chet Ramey) рекомендуется следующая команда:

```
complete -A helptopic help
```

С ее помощью ряд возможных вариантов автозавершения команды `help` ограничивается разделами справки. Выполнив эту команду, следует далее ввести команду `help` и два знака табуляции, чтобы оболочка отобразила список разделов справки.

`compopt` **Вывести или изменить параметры автозавершения команды**

```
compopt [-DE] [-o параметры] [+o параметры]
        [команды ...]
```

В отсутствие параметров вывести параметры автозавершения одной или больше заданных *команд* или же параметры совершаемого в настоящий момент автозавершения, если команда `compopt` вызывается без заданных *команд*. А при наличии параметров видоизменить существующие спецификации автозавершения заданных *команд* или же совершаемого в настоящий момент автозавершения, если команда `compopt` вызывается без заданных *команд*.

Параметры

-D Применить заданные ***параметры*** к "стандартному" автозавершению, совершаемому по умолчанию

-E Применить заданные ***параметры*** к "пустому" автозавершению

-o *параметр*

Активизировать заданный ***параметр***, относящийся к числу достоверных параметров команды **`complete`**

+o *параметр*

Дезактивизировать заданный ***параметр***, относящийся к числу достоверных параметров команды **`complete`**

`continue` **Пропустить остальную часть тела одного или более цикла**

```
continue [n]
```

Пропустить остальные команды в теле цикла `for`, `while`, `select` или `until`, возобновив выполнение цикла со следующего его шага (или пропустив *n* вложенных циклов).

`declare` **Объявить переменные и манипулировать их атрибутами**

`declare [`*параметры*`] [`*имя*`[=`*значение*`]]`

Объявить переменные оболочки и манипулировать их атрибутами. В теле функций переменные считаются локальными, как будто они объявлены с помощью команды `local`. Сначала должны быть заданы все *параметры*. См. также приведенное далее описание команды `typeset`.

Параметры

- **`-a`** Каждое указанное ***имя*** является индексированным массивом
- **`-A`** Каждое указанное ***имя*** является ассоциативным массивом
- **`-f`** Каждое указанное ***имя*** является функцией
- **`-F`** Для функций вывести только их имена и атрибуты, но не определение (т.е. тело функции)
- **`-g`** Если данная команда применяется в теле функции, то объявить переменную в глобальной, а не в локальной области видимости
- **`-i`** Каждая переменная считается целочисленной; в операции присваивания ее значение вычисляется как арифметическое выражение
- **`-l`** Пометить указанные ***имена*** как переменные, имеющие значения, приведенные к нижнему регистру букв после присваивания
- **`-n`** Каждое указанное ***имя*** является ссылкой на косвенную переменную. См. выше раздел "Косвенные переменные"
- **`-p`** В отсутствие указанных ***имен*** вывести все переменные, их значения и атрибуты. А при наличии указанных ***имен*** вывести имена, атрибуты и значения, если таковые установлены, заданных переменных. Если задан параметр **`-f`**, вывести определения функций
- **`-r`** Пометить указанные ***имена*** как доступные только для чтения. Все последующие операции присваивания завершатся неудачно. Кроме того, доступные только для чтения переменные нельзя устанавливать в исходное состояние
- **`-t`** Присвоить атрибут **`trace`** каждому указанному ***имени***. Трассируемые функции наследуют прерывание **DEBUG**. Данный атрибут не имеет никакого специального назначения для переменных
- **`-u`** Пометить указанные ***имена*** как переменные, имеющие значения, приведенные к верхнему регистру букв после присваивания
- **`-x`** Пометить указанные ***имена*** для экспорта в рабочую среду порожденных процессов

Если перечисленные выше параметры указаны со знаком **+**, а не **-**, то соответствующий атрибут запрещается. В отсутствие имен переменных все переменные, имеющие заданные атрибуты, выводятся в форме, допускающей их повторное чтение как входных данных для оболочки.

Примеры

```
$ declare -i val              # Объявить переменную val
                              # как целочисленную
$ val=4+7                     # Вычислить значение данной
                              # переменной
$ echo $val                   # Показать полученный результат
11

$ declare -r z=42             # Объявить переменную z как
                              # доступную только для чтения
$ z=31                        # Попытаться присвоить ей
                              # значение
bash: z: readonly variable    # Присваивание не удалось

$ echo $z
42
$ declare -p val z            # Показать атрибуты
                              и значения переменных
declare -i val="11"
declare -r z="42"
```

---

dirs **Вывести стек каталогов или манипулировать им**

dirs [-clpv] [+*n*] [-*n*]

Вывести стек каталогов, которым можно манипулировать с помощью команд pushd и popd.

Параметры

+n	Вывести *n*-й элемент слева; первый элемент считается нулевым
-n	Вывести *n*-й элемент справа; первый элемент считается нулевым
-c	Очистить стек каталогов (т.е. удалить из него все элементы)
-l	Получить более длинный список, где содержимое переменной **$HOME** не заменяется знаком ~
-p	Вывести стек каталогов по одному элементу в каждой строке
-v	Вывести стек каталогов по одному элементу в каждой строке, предварив каждый элемент его индексом в стеке

`disown` **Прекратить управление одним или большим числом заданий**

`disown [-ahr] [`*`задание`*` ...]`

Удалить одно или больше указанных *заданий* из списка заданий, который ведется в оболочке Bash. Конкретное *задание* обозначается как спецификация задания или идентификатор процесса.

Параметры

- **`-a`** Удалить все задания. Если задан параметр **`-h`**, пометить все задания
- **`-h`** Пометить известные задания вместо их удаления из списка, чтобы не получать сигнал **`SIGHUP`**, как пояснялось в разделе "Выполнение команд"
- **`-r`** В отсутствие указанных заданий удалить (или пометить) только выполняющиеся задания

`do` **Зарезервированное слово, с которого начинается тело цикла**

`do`

Это зарезервированное слово, предшествующее последовательности команд в теле цикла, определяемого оператором `for`, `while`, `until` или `select`.

`done` **Зарезервированное слово, которым завершается тело цикла**

Это зарезервированное слово, завершающее тело цикла, определяемого оператором `for`, `while`, `until` или `select`.

`echo` **Направить аргументы командной строки в стандартный вывод**

`echo [-eEn] [`*`строка`*`]`

Направить заданную *строку* в стандартный вывод. Это встроенная версия команды. Вполне возможно, что в вашей системе имеется самостоятельная исполняемая программа под тем же самым именем `echo`. Сведения о ней см. на оперативной странице *echo*(1) руководства своей системы.

Параметры

Если установлен параметр оболочки `xpg_echo` наряду с режимом работы по стандарту POSIX (с помощью команды `set`

-o posix), то команда echo не интерпретирует ни один из перечисленных ниже параметров.

- **-e** Разрешить интерпретацию управляющих последовательностей символов, как описано ранее в разделе "Управляющие последовательности символов". Они должны быть заключены в кавычки (или экранированы знаком ****), чтобы предотвратить их интерпретацию оболочкой
- **-E** Не интерпретировать управляющие последовательности символов — даже в тех системах, где по умолчанию команда **echo** должна их интерпретировать
- **-n** Не выводить завершающий знак новой строки

Примеры

```
$ echo "testing printer" | lpr
$ echo -e "Warning: ringing bell \a"
```

---

enable **Разрешить или запретить команды, встроенные в оболочку**

enable [-adnps] [-f *файл*] [*команда* ...]

Разрешить или запретить команды, встроенные в оболочку. Запрет встроенной команды дает возможность воспользоваться ее внешней версией. Ведь иначе будет использована ее встроенная версия, как, например, для команды echo или test.

Параметры

- **-a** Вывести сведения о всех встроенных командах: как запрещенных, так и разрешенных. Применяется вместе с параметром **-p**
- **-d** Удалить встроенную команду, загруженную ранее с помощью параметра **-f**
- **-f *файл***

  Загрузить новую встроенную команду из указанного ***файла*** совместно используемой библиотеки. Оболочка ищет нужный ***файл*** в каталогах, указанных в переменной **$BASH_LOADABLES_PATH**
- **-n** Запретить указанные встроенные команды
- **-p** Вывести список разрешенных встроенных команд
- **-s** Вывести только специальные встроенные команды по стандарту POSIX. В сочетании с параметром **-f** данный параметр определяет новую встроенную команду как специальную по стандарту POSIX

`esac` Зарезервированное слово, завершающее оператор выбора `case`

```
esac
```

Зарезервированное слово, которым завершается оператор выбора `case`.

`eval` Пересмотреть и выполнить уже обработанную введенную строку

`eval` *аргументы*

Как правило, команда `eval` применяется в сценариях оболочки, а в качестве ее *аргументов* указывается строка кода, содержащая переменные оболочки. Сначала команда `eval` принудительно подставляет переменные, а затем выполняет получающуюся в итоге команду. Такой "двойной просмотр" строки кода оказывается полезным всякий раз, когда переменные оболочки содержат знаки переадресации ввода-вывода, псевдонимы или другие переменные оболочки. (Например, переадресация ввода-вывода обычно происходит прежде подстановки переменных, и поэтому переменная, содержащая знаки переадресации ввода-вывода, должна быть сначала подставлена с помощью команды `eval`. В противном случае знаки переадресации ввода-вывода останутся неинтерпретированными.)

Пример

В следующем фрагменте кода из сценария оболочки демонстрируется, каким образом в команде `eval` конструируется правильно интерпретируемая команда:

```
for option
do
    # Определить, куда выводить результаты
    case "$option" in
    save)   out=' > $newfile' ;;
    show)   out=' | more' ;;
    esac
done

eval sort $file $out
```

exec **Заменить текущий сценарий оболочки или манипулировать дескрипторами ее файлов**

exec [*команда аргументы* ...]
exec [-a *имя*] [-cl] [*команда аргументы* ...]
exec *переадресации* ...

Выполнить заданную *команду* вместо текущего процесса, чтобы не создавать новый процесс. Если в команде exec указаны только *переадресации* (подробнее об этом см. выше в подразделе "Переадресация ввода-вывода с использованием дескрипторов файлов"), то ею удобно воспользоваться для открытия, закрытия, копирования или перемещения файлов по их дескрипторам, не прерывая выполнение сценария.

Параметры

- **-a** Использовать указанное ***имя*** в качестве первого аргумента (**argv[0]**) заданной ***команды***
- **-c** Очистить рабочую среду перед выполнением программы
- **-l** Предварить знаком "минус" первый аргумент (**argv[0]**) заданной ***команды***, как описано на оперативной странице руководства *login*(1)

Примеры

```
trap 'exec 2>&-' 0          # Закрыть стандартный вывод ошибок
stderr
                            # при выходе из сценария (по сигналу
0)

$ exec /bin/csh             # Заменить оболочку на C shell
                            # (неудачная мысль)
$ exec < infile             # Переназначить стандартный ввод
                            # в заданный файл infile
```

exit **Завершить сценарий оболочки**

exit [*n*]

Завершить сценарий оболочки с кодом завершения *n* (например, exit 1), где *n* может быть нулевым (удачным) или ненулевым (неудачным) кодом завершения. Если аргумент *n* не задан, оболочка возвращает код завершения самой последней выполнявшейся команды. Команда exit может быть

выдана из командной строки с целью закрыть окно (выйти из системы). Коды завершения могут изменяться в пределах от **0** до **255**. Любое прерывание по сигналу `EXIT` выполняется перед выходом из оболочки. При выходе из неинтерактивных оболочек выполняется файл `~/.bash_logout`. См. также выше раздел “Код завершения команды”.

Пример

```
if [ $# -eq 0 ]
then
    echo "Usage: $0 [-c] [-d] file(s)" 1>&2
    exit 1               # Ошибочный код завершения
fi
```

---

export **Экспортировать элементы или вывести сведения о них**

```
export [переменные]
export [имя=[значение] ...]
export -p
export [-fn] [имя=[значение] ...]
```

Передать (экспортировать) значение одной или нескольких *переменных* оболочки, придав им глобальное назначение, поскольку по умолчанию они являются локальными. Например, переменная, определенная в одном сценарии оболочки, должна быть экспортирована, если ее значение используется в других программах, вызываемых из данного сценария. Вторая форма команды `export` является ее версией по стандарту POSIX и аналогична первой форме, за исключением того, что в ней можно установить *значение* переменной по ее указанному *имени*, прежде чем экспортировать ее. По команде `export` можно экспортировать и функции.

Параметры

-f Имена для обращения к функциям, которые экспортируются в рабочую среду

-n Удалить указанные по имени переменные или функции из рабочей среды

-p Выполнить команду **declare -x**, прежде чем выводить имена и значения экспортируемых переменных. Это дает возможность сохранить список экспортируемых переменных для повторного чтения в дальнейшем. Выводить только имена экспортируемых функций, но не их тела

Примеры

Для экспорта переменной в первоначальной оболочке Bourne shell приходилось вводить две строки:

```
TERM=vt100
export TERM
```

А в оболочке Bash достаточно ввести одну строку:

```
export TERM=vt100
```

---

false — **Завершить с ложным возвращаемым значением**

```
false
```

Встроенная команда, завершающаяся ложным возвращаемым значением, обозначающим неудачный исход.

---

fc — **Вести предысторию выполнения команд**

fc [*параметры*] [*первая* [*последняя*]]
fc -e - [*прежняя*=*новая*] [*команда*]
fc -s [*прежняя*=*новая*] [*команда*]

Отобразить команды из списка предыстории их выполнения. (Использовать только один из параметров **-e**, **-l** или **-s**.)

Аргументы *первая* и *последняя* указываются как числа или символьные строки, обозначающие пределы отображаемых или редактируемых команд. Если аргумент *последняя* опущен, команда fc применяет единственную команду, обозначаемую аргументом *первая*. А если опущены оба аргумента, *первая* и *последняя*, команда fc редактирует предыдущую команду или перечисляет последние 16 выполнявшихся команд. Если же аргумент *первая* равен **-0**, это относится к текущей команде.

Во второй форме команда fc принимает указанную *команду* из предыстории, заменяет *прежнюю* команду на *новую* и выполняет видоизмененную команду. Если ни аргумент *прежняя*, ни *новая* не указан, то указанная *команда* выполняется повторно. А если не указана и *команда*, то повторно выполняется предыдущая команда. Аргумент *команда* указывается как

число или символьная строка подобно остальным аргументам. Третья форма команды `fc` аналогична второй ее форме. Примеры применения команды `fc` см. выше в соответствующем подразделе раздела "Предыстория выполнения команд".

Параметры

-e ***редактор***

Вызвать указанный ***редактор*** для редактирования заданных команд из предыстории их выполнения. Наименование редактора следует указывать вместе с параметром **-e**. В отсутствие параметра **-e** команда **fc** вызовет редактор, устанавливаемый по умолчанию в переменной **FCEDIT**. Если же эта переменна не установлена, оболочка Bash попытается обратиться к переменной **$EDITOR**. Если и она не установлена, то по умолчанию вызывается редактор **vi**. Начиная с версии 3.1 по умолчанию вызывается редактор **ed**, если оболочка работает в режиме по стандарту POSIX

-e - Выполнить (или повторить) команду из предыстории; см. приведенную выше вторую форму команды **fc**

-l Перечислить указанную команду, ряд команд или последние 16 выполнявшихся команд

-n Подавить нумерацию команд, перечисляемых с помощью параметра **-l**

-r Изменить на обратный порядок перечисления команд с помощью параметра **-l**

-s То же, что и **-e -**

---

fg **Перенести выполняющееся или приостановленное задание в приоритетный режим**

`fg [`*`идентификаторы заданий`*`]`

Перенести текущее задание или несколько заданий, имеющих указанные *идентификаторы заданий*, на выполнение в приоритетный режим. Подробнее об этом см. выше в разделе "Управление заданиями".

---

fi Зарезервированное слово, завершающее условный оператор if

`fi`

Зарезервированное слово, которым завершается условный оператор `if`.

```
for x [in [список]]
do
    команды
done
```

Выполнить заданные *команды* для переменной x (в дополнительном *списке* значений). Если *список* в условии in опущен, то предполагается список позиционных параметров ("$@"). А если подстановка *списка* оказывается пустой, то ни одна из заданных *команд* не выполняется.

Примеры

Вывести постранично файлы, указанные в командной строке, сохранив в каждом случае полученный результат:

```
for file
do
    pr $file > $file.tmp
done
```

То же, что и выше, но перенести выполнение всего цикла в фоновый режим:

```
for file
do
    pr $file > $file.tmp
done &
```

Найти главы с заданном списке слов (аналогично команде fgrep -f):

```
for item in $(cat program_list)
do
    echo "Checking chapters for"
    echo "references to program $item..."
    grep -c "$item.[co]" chap*
done
```

Извлечь однословное название из каждого файла, имя которого указано в командной строке, и воспользоваться им в качестве нового имени файла:

```
for file
do
    name=$(sed -n 's/NAME: //p' $file)
    mv $file $name
done
```

---

for **Выполнить цикл арифметических операций**

```
for (( начальное значение; условие; приращение ))
do
    команды
done
```

Цикл арифметических операций `for` подобен аналогичному циклу в языке C. В этом цикле сначала вычисляется *начальное значение*, а тело данного цикла выполняется до тех пор, пока истинно заданное *условие*. Любое из арифметических выражений в начале данной разновидности цикла `for` может быть опущено. В частности, отсутствующее *условие* интерпретируется как истинное.

Пример

Найти указанную фразу в каждой нечетной главе:

```
for ((x=1; x <= 20; x += 2))
do
    grep $1 chap$x
done
```

---

function **Определить функцию оболочки**

```
function имя { команды; } [переадресации]
function имя () { команды; } [переадресации]
```

Определить функцию оболочки по заданному *имени*. Подробнее о семантике функций см. выше в разделе “Функции”.

Пример

Определить функцию для подсчета файлов:

```
$ function countfiles {
> ls | wc -l
> }
```

getopts — Обработать параметры и аргументы командной строки

```
getopts строка имя [аргументы]
```

Обработать *аргументы*, если они указаны, и проверить наличие допустимых параметров в командной строке. Команда `getopts` применяется в сценариях оболочки и призвана обеспечить стандартный синтаксис параметров командной строки.

Стандартный синтаксис требует, чтобы параметры командной строки начинались со знака **-**. Параметры можно указывать подряд после единственного знака **-**, а окончание их обработки в командной строке — двумя знаками **--**. Заданная *строка* содержит буквы параметров, которые должны быть распознаны командой `getopts` при выполнении сценария оболочки. Достоверные параметры обрабатываются по очереди и сохраняются в переменной оболочки, имеющей указанное *имя*. Если же эта переменная доступна только для чтения, то команда `getopts` завершится с возвращаемым значением **2**.

Если после символа, обозначающего параметр в заданной *строке*, следует двоеточие, то после самого параметра должен следовать один аргумент или больше. (Несколько аргументов должны быть указаны как одно *слово* оболочки. С этой целью аргументы заключаются в кавычки или разделяются запятыми. Сценарий следует писать, принимая во внимание именно такую форму указания нескольких аргументов.)

В команде `getopts` применяются переменные `OPTARG`, `OPTIND` и `OPTERR`. (Более подробно о них см. в разделе "Встроенные в оболочку переменные".)

hash — Вести таблицу обнаруженных ранее команд

```
hash [-dlrt] [команды]
hash [-p файл] [команды]
```

По мере обнаружения команд в ходе поиска по пути, заданному в переменной окружения `$PATH`, оболочка запоминает обнаруженные места нахождения команд во внутренней хеш-таблице. А при последующем вводе команды оболочка выбирает соответствующее сохраненное значение из этой хеш-таблицы.

В отсутствие аргументов команда hash перечисляет текущий набор хешированных команд. При этом показываются *обращения*, обозначающие, сколько раз команда вызывалась из оболочки, а также наименование команды. Если таблица пуста, а оболочка работает в режиме по стандарту POSIX, то команда hash ничего не выводит. В противном случае она направляет сообщение "hash: hash table empty" (команда hash: хеш-таблица пуста) в стандартный вывод.

Если заданы *команды*, оболочка вводит их в хеш-таблицу. Если же *команды* заданы без параметров, оболочка устанавливает в исходное нулевое состояние "подсчет обращений", связанных с каждой командой.

Переменная массива BASH_CMDS обеспечивает программный доступ ко всем записям в хеш-таблице. Более подробно о ней см. в разделе "Переменные, встроенные в оболочку".

Параметры

-d Удалить из хеш-таблицы только указанные команды

-l Вывести результат в такой форме, чтобы его можно было повторно прочитать при перестройке хеш-таблицы

-p *файл*

Связать указанный ***файл*** с заданной ***командой*** в хеш-таблице

-r Удалить из хеш-таблицы все команды

-t Если указано лишь одно наименование команды, вывести полный путь к ней. А если указано несколько наименований, то вывести наименование команды и полный путь к ней в двух столбцах

Совет

Хеш-таблицу можно очистить не только с помощью параметра **-r**, но и в тот момент, когда присваивается значение переменной окружения $PATH. В частности, чтобы очистить хеш-таблицу, не оказывая влияния на путь поиска команд, достаточно выполнить операцию присваивания PATH=$PATH. Это наиболее удобно, если новая версия команды установлена в том каталоге, который был указан в переменной окружения $PATH раньше, чем текущая версия данной команды.

help — **Вывести сведения об использовании команды**

`help [-dms] [`*шаблон*`]`

Направить в стандартный вывод сведения об использовании каждой команды, совпадающей с заданным *шаблоном*. В эти сведения включается описание каждого параметра команды.

Параметры

- **-d** Вывести краткое описание назначения команды
- **-m** Вывести полное описание назначения команды в такой же форме, как и на оперативной странице руководства по системе Unix
- **-s** Вывести краткие сведения об использовании команды

Примеры

```
$ help -s cd                          # Краткая справка
cd: cd [-L|[-P [-e]] [-@]] [dir]

$ help true                           # Полная справка
true: true
      Return a successful result.     # Возвратить удачный
                                      # результат
      Exit Status:                    # Код завершения
      Always succeeds.                # Всегда удачный исход
```

history — **Вывести предысторию выполнения команд**

`history [`*подсчет*`]`
`history [`*параметры*`]`

Вывести команды из списка предыстории их выполнения или манипулировать файлом предыстории. А в отсутствие заданных параметров или аргументов отобразить список предыстории с номерами выполнявшихся команд. Если указан аргумент *подсчет*, вывести лишь заданное в нем количество самых последних выполнявшихся команд.

Оболочка Bash сохраняет предысторию выполнения команд для любой оболочки, где такая предыстория активизирована (по команде `-o history` и при установке переменной `HISTFILE`), а не только для интерактивных оболочек.

Параметры

-a Добавить в файл предыстории новые строки с командами, которые были выполнены с момента последней записи в данный файл. Запись происходит немедленно

-c Очистить список предыстории, удалив из него все элементы

-d *позиция*

Удалить из списка предыстории элемент, находящийся на указанной ***позиции***

-n Прочитать непрочитанные строки из файла предыстории и добавить их в список предыстории

-p *аргумент*

Выполнить подстановку предыстории в стиле оболочки **csh** по каждому указанному ***аргументу***, направив полученные результаты в стандартный вывод. Эти результаты не сохраняются в списке предыстории

-r Прочитать содержимое файла предыстории и заменить им список предыстории

-s *аргумент*

Сохранить указанные ***аргументы*** в списке предыстории как единую запись

-w Записать текущий список предыстории в файл предыстории, переписав его полностью. Запись происходит немедленно

Совет

С любезного разрешения Эли Зарецкого (Eli Zaretskii) рекомендуется следующая команда:

```
export PROMPT_COMMAND='history -a'
```

Эта команда гарантирует, что файл предыстории будет всегда актуальным, даже если пользователь не выйдет из системы корректно, а сеанс его работы останется активным.

---

if **Синтаксис для условного оператора if-then-else**

```
if условие1
then команды1
[ elif условие2
  then команды2 ]
     .
     .
     .
[ else команды3 ]
fi
```

Если удовлетворяется заданное *условие*$_1$, выполнить указанные *команды*$_1$. Если же удовлетворяется заданное *условие*$_2$, выполнить указанные *команды*$_2$. А если не удовлетворяется ни одно из заданных условий, то выполнить указанные *команды*$_3$. Зачастую условия задаются с помощью команд `test` и `[[ ]]`. Полный перечень задаваемых условий приведен далее в описании команды `test`, а дополнительные примеры их применения — ранее в описании команд `[[ ]]`, `:` и `exit`.

Примеры

Предварить нулем числа меньше **10**:

```
if [ $counter -lt 10 ]
then number=0$counter
else number=$counter
fi
```

Создать каталог, если он отсутствует:

```
if ! [ -d $dir ]
then
    mkdir -m 775 $dir
fi
```

`jobs` Перечислить выполняющиеся или приостановленные задания

`jobs [`*параметры*`] [`*идентификаторы заданий*`]`

Перечислить все выполняющиеся или приостановленные задания или же те из них, на которые указывают *идентификаторы заданий*. С помощью данной команды можно, например, проверить, выполняется ли по-прежнему длительное задание на компиляцию или форматирование текста. Ею удобно также воспользоваться перед выходом из системы. См. также раздел “Управление заданиями”.

Параметры

-l Перечислить задания по их идентификаторам, а процессы по их групповым идентификаторам

-n Перечислить только те задания, состояние которых не изменилось с момента последнего уведомления

-p	Перечислить только процессы по их групповым идентификаторам
-r	Перечислить только выполняющиеся задания
-s	Перечислить только приостановленные задания

-x *команда*

Заменить каждый идентификатор задания, обнаруженный в указанной ***команде***, связанным с ним идентификатором процесса, а затем выполнить указанную ***команду***

kill **Послать сигнал прерывания одному или нескольким заданиям**

kill [*параметры*] *идентификаторы*

Прервать каждый процесс или задание по указанным *идентификаторам*. Для этого нужно быть владельцем процесса или привилегированным пользователем. Эта встроенная команда аналогична внешней команде kill, но она позволяет также указывать символические имена заданий. Неподатливые процессы могут, как правило, быть уничтожены по сигналу **9**. См. также раздел "Управление заданиями".

Команда kill -l выводит список имеющихся наименований сигналов. Этот список может меняться в зависимости от конкретной архитектуры системы.

Сигналы и их номера определяются в заголовочном файле <signal.h> на языке C. В этот файл могут быть включены и другие сигналы, а следовательно, конкретное местоположение определений сигналов может меняться в разных системах.

Параметры

-l [*n*]	В отсутствие аргумента ***n*** перечислить наименования сигналов. Числовое значение аргумента ***n*** интерпретируется как номер сигнала или же как код завершения процесса, прерываемого по сигналу (**128** + ***n***). Но в любом случае команда **kill** выводит соответствующее наименование сигнала
-L [*n*]	То же, что и параметр **-l**
-n *номер*	Послать сигнал, имеющий указанный ***номер***
-s *имя*	Послать сигнал, имеющий указанное ***имя***
- *сигнал*	Послать сигнал, имеющий номер, заданный в заголовочном файле **<signal.h>**, или наименование, определяемое по команде **kill -l**. Если посылается сигнал номер **9**, процесс или задание уничтожается безусловно

В режиме работы по стандарту POSIX необходимо отбросить префикс SIG в наименованиях сигналов. А по команде `kill -l` наименования всех сигналов выводятся в одной строке.

---

let **Выполнить арифметическую операцию**

```
let выражения
(( выражения ))
```

Выполнить арифметическую операцию, указанную в одном или нескольких *выражениях*, состоящих из чисел, операций и переменных оболочки, которые совсем не обязательно предварять знаком **$**. Выражения должны быть заключены в кавычки, если они содержат пробелы или другие специальные символы, а в форме ((...)) это делается автоматически. Дополнительные сведения и примеры применения данной команды см. в разделе "Арифметические выражения", а также на оперативной странице руководства *expr*(1). Если вычисление окончательного выражения дает нулевой результат, команда `let` возвращает единичный код завершения (неудачный исход), в противном случае — нулевой код завершения (удачный исход).

Примеры

В каждом из приведенных ниже примеров значение переменной `i` увеличивается на единицу.

```
i=`expr $i + 1`          # Во всех оболочках типа Bourne
shell
let i=i+1                # В оболочке Bash
let "i = i + 1"
(( i = i + 1 ))
(( i += 1 ))
(( i++ ))
if (( i % 2 == 0 )) ...
```

---

local **Объявить локальные переменные в функциях оболочки**

```
local [параметры] [имя[=значение]]
```

Объявить локальные переменные для применения в теле функций. Задаваемые *параметры* те же, что и для команды `declare` (полный их перечень см. выше, в описании команды

declare). Пользоваться командой local вне тела функции считается ошибкой. Если *имя* указано как **-**, оболочка сохранит все значения однобуквенных параметров и восстановит их после возврата из функции.

logout **Выйти из оболочки**

```
logout
```

Выйти из исходной оболочки. При выходе из нее выполнить файл ~/.bash_logout. Данная команда завершится неудачно, если текущая оболочка не является исходной.

mapfile **Прочитать содержимое файла в массив оболочки**

mapfile [*параметры*] [*массив*]

Прочитать стандартный ввод в заданный *массив* по одной строке на каждый его элемент. Если *массив* не задан, использовать значение переменной MAPFILE. С помощью параметра **-u** может быть задан альтернативный дескриптор файла.

Параметры

-c *подсчет*

Обозначает количество строк для параметра **-C**. По умолчанию принимает значение **5000**

-C *команда*

Для каждого количества строк вычислить указанную ***команду***, передав ей индекс заданного ***массива*** и присваиваемую строку. Количество строк устанавливается с помощью параметра **-c**. А данный параметр применяется, главным образом, в отладчике оболочки Bash

-d *разделитель*

Использовать первый символ указанного ***разделителя*** вместо знака новой строки для окончания вводимых строк. Если указанный ***разделитель*** оказывается нулевой строкой, использовать в качестве разделителя нулевой байт (NUL в коде ASCII)

-n *подсчет*

Прочитать хотя бы столько строк, сколько определяет заданный ***подсчет***. Если же задан нулевой ***подсчет***, прочитать все строки

-O *индекс*

Заполнить заданный ***массив***, начиная с указанного первоначального ***индекса***, который по умолчанию равен нулю

-s *подсчет*

Пропустить (т.е. проигнорировать) первые строки, количество которых определяет заданный ***подсчет***

-t Удалить конечный разделитель (обычно знак новой строки) из каждой прочитанной строки

-u *n*

Читать строки из файла по заданному дескриптору ***n***, а не из стандартного ввода

---

popd — Извлечь каталог из стека каталогов

`popd [-n] [+отсчет] [-отсчет]`

Извлечь каталог из вершины стека каталогов, отображаемого по команде `dirs`, а затем перейти к новому каталогу на вершине стека или манипулировать стеком каталогов.

Параметры

+*отсчет*

Удалить элемент, находящийся слева в списке, отображаемом по команде **dirs**, начиная заданный ***отсчет*** с нуля. При этом смена каталога не происходит

-*отсчет*

Удалить элемент, находящийся справа в списке, отображаемом по команде **dirs**, начиная заданный ***отсчет*** с нуля. При этом смена каталога не происходит

-n Не переходить к новому каталогу на вершине стека, а только манипулировать стеком

---

printf — Произвести форматированный вывод аргументов командной строки

`printf [-v переменная] формат [значение ...]`

Произвести форматированный вывод аналогично функции `printf()` языка C по стандарту ANSI, как поясняется на оперативной странице руководства *printf*(3). Управляющие последовательности символов в заданном *формате* экранируются, как пояснялось ранее в разделе "Управляющие последовательности символов". Строка форматирования повторно используется с самого начала, если задано больше значений, чем указано в ней спецификаторов формата.

Параметр

-v *переменная*

Сохранить полученный результат в заданной ***переменной***, а не направлять его в стандартный вывод, даже если результат окажется пустым. Заданная ***переменная*** может быть элементом массива

Дополнительные спецификаторы формата

В оболочке Bash допускаются следующие дополнительные спецификаторы формата.

%b Подставить управляющие последовательности символов в символьные строки с аргументами. К их числу относятся управляющие последовательности символов, воспринимаемые в команде **echo -e** (см. также раздел "Управляющие последовательности символов")

%q Вывести заключенную в кавычки символьную строку для последующего ее повторного чтения

%(*формат_даты*)T

Вывести величины времени, используя заданный ***формат_даты*** в качестве строки форматирования, как поясняется на оперативной странице руководства *strftime*(3). Использовать текущее время в отсутствие заданных аргументов

---

pushd **Поместить каталог в стек каталогов**

```
pushd [-n] [каталог]
pushd [-n] [+отсчет] [-отсчет]
```

Ввести указанный `каталог` в стек каталогов или произвести циклический сдвиг стека каталогов. В отсутствие аргументов поменять местами две верхние записи на вершине стека, заменив прежнюю запись новой.

Параметры

+*отсчет*

Произвести циклический сдвиг стека каталогов, чтобы переместить на вершину стека элемент, находящийся слева в списке, отображаемом по команде **dirs**, начиная заданный ***отсчет*** с нуля

-*отсчет*

Произвести циклический сдвиг стека каталогов, чтобы переместить на вершину стека элемент, находящийся справа в списке, отображаемом по команде **dirs**, начиная заданный ***отсчет*** с нуля

-n Не переходить к новому каталогу на вершине стека, а только манипулировать стеком

pwd **Вывести рабочий каталог**

```
pwd [-LP]
```

Направить текущий рабочий каталог в стандартный вывод. Завершить команду с кодом неудачного завершения, если переменная `PWD` доступна только для чтения.

Параметры

Параметры данной команды позволяют задавать логическую или физическую интерпретацию выводимого пути к рабочему каталогу. См. также приведенное ранее описание команды `cd`.

-L Использовать логический путь (т.е. то, что введено пользователем, включая любые символические ссылки) и значение переменной **PWD** для обращения к текущему каталогу. Это делается по умолчанию

-P Использовать физический путь к текущему каталогу из файловой системы

read **Прочитать данные, сохранив их в одной или нескольких переменных оболочки**

read [*параметры*] [*переменная*$_1$ [*переменная*$_2$ ...]]

Прочитать одну строку из стандартного ввода и присвоить каждое слово соответствующей *переменной*, а все оставшиеся слова — последней переменной. Если указана только одна переменная, ей присваивается вся прочитанная строка. Оболочка Bash удаляет нулевые (NUL в коде ASCII) байты из входных данных. Соответствующие примеры см. ниже, а также в описании команды `case`. Команда `read` возвращает нулевой код завершения, если только не будет достигнут признак *конца файла*. Если переменные не заданы, входные данные сохраняются в переменной `REPLY`. В режиме работы оболочки по стандарту POSIX перехватываемые сигналы могут прервать выполнение команды `read`, и в этом случае она возвратит значение **128** + номер сигнала, частично отвергнув прочитанные входные данные.

Параметры

`-a` ***массив***

Прочитать входные данные в указанный индексированный ***массив***. Сообщить об ошибке, если указанный ***массив*** существует и является ассоциативным

`-d` ***разделитель***

Прочитать входные данные вплоть до первого появления указанного ***разделителя*** вместо знака новой строки. Если указанный ***разделитель*** окажется нулевой строкой, использовать в качестве разделителя нулевой байт (NUL в коде ASCII)

`-e` Использовать библиотеку `readline`, если входные данные читаются с терминала

`-i` ***текст***

Если используется библиотека `readline`, поместить указанный ***текст*** во внутренний буфер редактирования

`-n` ***количество***

Прочитать хотя бы заданное ***количество*** байтов. Если символ разделителя встретится прежде, чем будет прочитано заданное ***количество*** байтов, прервать дальнейшее чтение входных данных

`-N` ***количество***

Прочитать хотя бы заданное ***количество*** байтов. Символы разделителей во входных данных не прерывают их чтение оболочкой Bash, а, напротив, включаются в прочитанные данные

`-p` ***приглашение***

Вывести заданное ***приглашение***, прежде чем читать входные данные

`-r` Режим без обработки входных данных; игнорировать знак `\` в качестве символа продолжения строки

`-s` Читать негласно, т.е. без эхоотображения входных данных на экране терминала

`-t` ***выдержка времени***

Возвратить `1`, если при чтении с терминала или из канала входные данные отсутствуют по истечении заданной ***выдержки времени***. Благодаря этому исключается зависание приложения в бесконечном ожидании входных данных от пользователя. Числовые значения ***выдержки времени*** могут быть дробными. Если задана нулевая ***выдержка времени***, а для чтения имеются входные данные, то команда `read` возвратит код удачного завершения. Входные данные, частично прочитанные до истечения заданной ***выдержки времени***, сохраняются в переменной, указанной по именем ***переменная***$_1$, а остальные переменные очищаются. Если же входные данные не были прочитаны до истечения заданной ***выдержки времени***, команда `read` возвратит код завершения больше `128`

`-u` `[`***n***`]`

Прочитать входные данные из файла с заданным дескриптором ***n***, по умолчанию равным нулю

Примеры

Прочитать входные данные, сохранив их в трех переменных:

```
$ read first last address
Sarah Caldwell 123 Main Street

$ echo -e "$last, $first\n$address"
Caldwell, Sarah
123 Main Street
```

Выдать приглашение на ввод двух показаний температуры:

```
$ read -p "High low: " n1 n2
High low: 65 33
```

---

readarray — **Прочитать содержимое файла в массив**

readarray [*параметры*] [*массив*]

Эта команда действует подобно команде mapfile. Подробнее см. приведенное ранее описание команды mapfile.

---

readonly — **Пометить переменные как доступные только для чтения**

readonly [-aAfp] [*переменная*[=*значение*] ...]

Предотвратить присваивание значений указанным переменным оболочки. Первоначальное значение переменной может быть задано с помощью синтаксиса присваивания, но впоследствии это значение не подлежит изменению. Переменные, доступные только для чтения, нельзя установить в исходное состояние.

Параметры

-a Каждая указанная ***переменная*** должна обозначать индексированный массив

-A Каждая указанная ***переменная*** должна обозначать ассоциативный массив

-f Каждая указанная ***переменная*** должна обозначать функцию

-p Вывести обозначение **readonly**, прежде чем выводить имена и значения переменных, доступных только для чтения. Это дает возможность сохранить список переменных, доступных для повторного чтения в дальнейшем

return **Возвратить код завершения из функции оболочки**

```
return [n]
```

Использовать данную команду в теле функции. Выйти из функции с кодом завершения *n* или же с кодом завершения выполнявшейся ранее команды. Если задается отрицательное значение *n*, предварить его знаками `--`.

select **Представить пункты меню для выполнения блока кода**

```
select x [in список]
do
    команды
done
```

Направить список пунктов меню в стандартный вывод ошибок, пронумеровав их в том порядке, в каком они представлены в указанном *списке*. Если условие `in` *список* не задано, пункты меню выбираются из командной строки (в форме `"$@"`). Сразу за меню следует строка приглашения (т.е. значение переменной `$PS3`). По приглашению из переменной `$PS3` пользователь выбирает пункт меню, вводя его номер, или повторно отображает меню, нажимая клавишу <Enter>. Введенные пользователем данные сохраняются в переменной оболочки `REPLY`. Если введен достоверный пункт меню, оболочка устанавливает выбранное значение в переменной цикла *x* и выполняет указанные *команды*. Цикл завершается вводом признака конца файла.

Пример

```
PS3="Select the item number: "
select event in Format Page View Exit
do
    case "$event" in
    Format)   nroff $file | lpr;;
    Page)     pr $file | lpr;;
    View)     more $file;;
    Exit)     exit 0;;
    * )       echo "Invalid selection";;
    esac
done
```

Из данного сценария выводится следующий результат:

```
1. Format
2. Page
3. View
4. Exit
Select the item number:
```

---

set — **Манипулировать параметрами оболочки и командной строки в сценарии**

set [*параметры аргумент$_1$ аргумент$_2$ ...*]

В отсутствие аргументов команда set выводит значения всех переменных, известных в текущей оболочке. Параметры могут быть активизированы (в форме -*параметр*) или дезактивизированы (в форме +*параметр*), а также установлены при вызове оболочки (подробнее об этом см. в разделе "Вызов оболочки"). Аргументы, не относящиеся к параметрам, присваиваются по порядку ($1, $2 и т.д.).

Параметры

-a Автоматически помечать переменные для экспорта после их определения или изменения

-b Выводить сообщения, как только задания будут завершены; не ожидать вплоть до следующего приглашения

-B Активизировать раскрытие скобок. Делается по умолчанию

-C Предотвратить перезапись посредством переадресации ввода-вывода **>**; использовать переадресацию ввода-вывода **>|** для перезаписи файлов

-e Завершить команду, если она выдает ненулевой код завершения. Перед выходом из оболочки выполняется прерывание **ERR**. Конкретное поведение замысловато, и поэтому оно дополнительно поясняется далее, в подразделе "Подробнее о команде **set -e**"

-E Обусловить наследование прерывания **ERR** в функциях оболочки, подстановках команд и подоболочках

-f Игнорировать метасимволы подстановки имен файлов (например, *****, **?**, **[ ]**)

-h Находить команды по мере их определения. Делается по умолчанию. См. также приведенное ранее описание команды **hash**

-H Активизировать подстановку предыстории в стиле оболочки **csh**. Делается по умолчанию. (Начиная с версии Bash 5.0, этот режим будет отключаться по умолчанию.) См. также раздел "Предыстория команд в стиле оболочки C shell"

-k Присваивать значения переменным окружения (в форме ***переменная=значение***) независимо от их местоположения в командной строке. Как правило, операции присваивания должны указываться прежде имени команды

-m Активизировать управление заданиями. Фоновые задания выполняются в отдельной группе процессов. Параметр **-m** зачастую устанавливается автоматически. Он активизируется в интерактивных оболочках, но не в сценариях

-n Читать команды, но не выполнять их, что удобно для проверки синтаксиса. В интерактивных оболочках этот параметр игнорируется

+o [*режим*]

Если задан конкретный ***режим***, отменить этот режим работы оболочки. По команде **set +o** выводятся настройки всех текущих режимов в такой форме, чтобы их можно было повторно прочитать в дальнейшем

-o [*режим*]

Перечислить все режимы работы оболочки или активизировать заданный ***режим***. Многие режимы работы оболочки могут быть также установлены с помощью других параметров данной команды. Эти режимы перечислены ниже:

allexport	То же, что и параметр **-a**
braceexpand	То же, что и параметр **-B**
emacs	Выбрать редактор **emacs** для редактирования командных строк
errexit	То же, что и параметр **-e**
errtrace	То же, что и параметр **-E**
functrace	То же, что и параметр **-T**
hashall	То же, что и параметр **-h**
histexpand	То же, что и параметр **-H**
history	Активизировать предысторию. Делается по умолчанию
ignoreeof	Не обрабатывать сигналы о достижении конца файла. Для выхода из оболочки следует ввести команду **exit**
keyword	То же, что и параметр **-k**
monitor	То же, что и параметр **-m**
noclobber	То же, что и параметр **-C**
noexec	То же, что и параметр **-n**
noglob	То же, что и параметр **-f**

nolog	Пропустить определения функций в файле предыстории. Игнорируется, хотя принято в оболочке Bash
notify	То же, что и параметр **-b**
nounset	То же, что и параметр **-u**
onecmd	То же, что и параметр **-t**
physical	То же, что и параметр **-P**
pipefail	Заменить код завершения конвейера на код завершения последней неудачно завершившейся команды или нулевой код завершения, если все команды в конвейере завершились удачно
posix	Перейти в режим работы по стандарту POSIX
privileged	То же, что и параметр **-p**
verbose	То же, что и параметр **-v**
vi	Выбрать редактор **vi** для редактирования командных строк
xtrace	То же, что и параметр **-x**

+p Заменить действующий идентификатор пользователя его реальным идентификатором

-p Начать работу с правами привилегированного пользователя. Не читать значения переменной окружения **$ENV** или **$BASH_ENV**, не импортировать функции из рабочей среды, а также игнорировать значения переменных окружения **$BASHOPTS**, **$CDPATH**, **$GLOBIGNORE** и **$SHELLOPTS**

-P Всегда использовать физические пути для выполнения команд **cd** и **pwd**

-t Завершить после выполнения одной команды

-T Обусловить наследование прерываний **DEBUG** и **RETURN** в функциях оболочки, подстановках команд и подоболочках

-u Интерпретировать при подстановках неустановленные переменные как ошибки. Тем не менее ссылки на переменные **$@** и **$*** не считаются ошибками в отсутствие позиционных параметров

-v Отображать каждую командную строку оболочки, когда она читается

-x Отображать команды вместе с их аргументами, когда они выполняются, предваряя их приглашением из переменной **PS4**. Этим обеспечивается пошаговая трассировка сценариев оболочки

- Отключить параметры **-x** и **-v**, а также обработку параметров. Входит в состав данной команды ради совместимости с прежними версиями оболочки Bash

-- Служит для обозначения последнего параметра, отключая обработку параметров, чтобы не путать с параметрами те аргументы, которые предваряются знаком **-**. (Например, в позиционном параметре **$1** можно установить значение **-1**.) Если аргументы после знаков **--** отсутствуют, то установить позиционные параметры в исходное состояние

Сравнительная таблица параметров

Ниже приведена сравнительная таблица параметров команды set.

Параметр	Равнозначный параметр
-a	-o allexport
-b	-o notify
-B	-o braceexpand
-C	-o noclobber
-e	-o errexit
-E	-o errtrace
-f	-o noglob
-h	-o hashall
-H	-o histexpand
-k	-o keyword
-m	-o monitor
-n	-o noexec
-o	allexport -a
-o	braceexpand -B
-o braceexpand	-B
-o emacs	
-o errexit	-e
-o errtrace	-E
-o functrace	-T
-o hashall	-h
-o history	
-o histexpand	-H
-o ignoreeof	
-o keyword	-k
-o monitor	-m
-o noclobber	-C
-o noexec	-n
-o noglob	-f
-o nolog	
-o notify	-b
-o nounset	-u
-o onecmd	-t
-o physical	-P
-o pipefail	

Параметр	Равнозначный параметр
`-o posix`	
`-o privileged`	`-p`
`-o verbose`	`-v`
`-o vi`	
`-o xtrace`	`-x`
`-p`	`-o privileged`
`-P`	`-o physical`
`-t`	`-o onecmd`
`-T`	`-o functrace`
`-u`	`-o nounset`
`-v`	`-o verbose`
`-x`	`-o xtrace`

Подробнее о команде `set -e`

Если по команде `set -e` установлен режим завершения, то выход из оболочки происходит при неудачном завершении конвейера, который может состоять из единственной команды; отдельной команды подоболочки, заключенной в круглые скобки; или же любой команды из группы, заключенной в фигурные скобки.

В режиме работы по стандарту POSIX оболочки, предназначенные для выполнения подстановок команд, наследуют режим завершения, устанавливаемый по команде `set -e`. В противном случае подобные оболочки наследуют данный режим в зависимости от установки параметра оболочки `inherit_errexit`.

Неудачное завершение команды (с ненулевым кодом) не приводит к выходу из оболочки в следующих случаях.

1. В списке, который следует после цикла `while` или `until`, имеется любая команда.
2. После условного оператора `if` или `elif` следует конвейер.
3. В логической операции **&&** или **||** имеется любая команда, кроме последней.
4. В конвейере имеется любая команда, кроме последней.
5. Назначение команды меняется на обратное с помощью знака `!`.

В общем, опытные программирующие на языке оболочки считают, что на практике команда `set -e` редко и едва ли вообще находит применение. Она существует лишь ради исторической совместимости, и поэтому вместо нее лучше тщательно программировать, чтобы отлавливать любые или все ошибки, которые могут возникнуть в принципе.

Примеры

```
set -- "$num" -20 -30   # Установить значение переменной $num
                        # в позиционном параметре $1,
                        # значение -20 в позиционном
                        # параметре $2,
                        # значение -30 в позиционном
                        # параметре $3
set -vx                 # Показать каждую команду дважды:
                        # один раз — при чтении, а другой
                        # раз — при выполнении
set +x                  # Остановить трассировку команд
set -o noclobber        # Запретить перезапись файлов
set +o noclobber        # Снова разрешить перезапись файлов
```

---

shift — **Сдвинуть аргументы командной строки влево**

`shift [`*n*`]`

Сдвинуть позиционные параметры (например, `$2` на место `$1`). Если задан аргумент *n*, то сдвиг происходит на *n* позиций. Данная команда нередко применяется в циклах `while` для перебора аргументов в командной строке.

Пример

```
shift $(($1 + $6)) # Использовать результат
                   # вычисления заданного
                   # выражения для отсчета
                   # сдвигаемых позиций
```

---

shopt — **Манипулировать параметрами оболочки**

`shopt [-opqsu] [`*параметр*`]`

Установить или сбросить параметры оболочки. Если параметры отсутствуют или указан только параметр **-p**, вывести

наименования и установки параметров. Более подробно различные параметры обсуждались в разделе “Параметры оболочки”.

Параметры

- **-o** Каждый заданный ***параметр*** должен относиться к тем, которые перечисляются в команде **set -o**, а не в разделе “Параметры оболочки”
- **-p** Вывести установки параметров в виде команд **shopt**, чтобы их можно было повторно прочитать в дальнейшем
- **-q** Негласный режим. Если заданный ***параметр*** установлен, то возвращается нулевой код завершения, а иначе — ненулевой код завершения. Если же задано несколько ***параметров***, то все они должны быть установлены для возврата нулевого кода завершения
- **-s** Установить заданные ***параметры***. В отсутствие заданных ***параметров*** вывести только те из них, которые установлены
- **-u** Сбросить заданные ***параметры***. В отсутствие заданных ***параметров*** вывести только те из них, которые сброшены

---

source — **Прочитать и выполнить файл в текущей оболочке**

source *файл* [аргументы]

Данная команда действует подобно команде `.` (точка). Подробнее см. приведенное ранее описание команды `.` (точка).

---

suspend — **Приостановить выполнение текущей оболочки**

suspend [-f]

Приостановить выполнение текущей оболочки. Данная команда нередко используется для прерывания команды `su`.

Параметр

- **-f** Принудительно приостановить выполнение оболочки, даже если она является исходной

---

test — **Вычислить условия, задаваемые в циклах и условных операторах**

test *условие*
[*условие*]
[[*условие*]]

Вычислить заданное *условие*, и если его значение окажется истинным, то возвратить нулевой код завершения, а иначе — ненулевой. В одной альтернативной форме данной команды вместо ключевого слова `test` применяются квадратные скобки (`[ ]`), а в другой — двойные квадратные скобки (`[[ ]]`). В последнем случае разделение слов и подстановка путей не выполняется (см. приведенное ранее описание команды `[[ ]]`). Заданное *условие* создается с помощью перечисленных ниже выражений. Условия считаются истинными, если верно их описание.

Условия проверки файлов

Ниже перечислены допустимые условия проверки файлов.

-a ***файл***	Заданный ***файл*** существует (не рекомендуется для применения; лучше воспользоваться параметром **-e**)
-b ***файл***	Заданный ***файл*** существует и является специальным файлом блочного устройства ввода-вывода
-c ***файл***	Заданный ***файл*** существует и является специальным файлом символьного устройства
-d ***файл***	Заданный ***файл*** существует и является каталогом
-e ***файл***	Заданный ***файл*** существует. То же, что и параметр **-a**, но существует ради совместимости со стандартом POSIX
-f ***файл***	Заданный ***файл*** существует и является обычным файлом
-g ***файл***	Заданный ***файл*** существует и установлен его бит смены идентификатора группы
-G ***файл***	Заданный ***файл*** существует и его группе присвоен действующий идентификатор группы
-h ***файл***	Заданный ***файл*** существует и является символической ссылкой. То же, что и параметр **-L**
-k ***файл***	Заданный ***файл*** существует и установлен его бит закрепления
-L ***файл***	Заданный ***файл*** существует и является символической ссылкой. То же, что и параметр **-h**
-N ***файл***	Заданный ***файл*** существует и видоизменен после его чтения в последний раз
-O ***файл***	Заданный ***файл*** существует и принадлежит пользователю с действующим идентификатором

`-p` ***файл***	Заданный ***файл*** существует и является именованным каналом обратного магазинного типа (FIFO)
`-r` ***файл***	Заданный ***файл*** существует и доступен для чтения
`-s` ***файл***	Заданный ***файл*** существует и его размер больше нуля
`-S` ***файл***	Заданный ***файл*** существует и является сокетом
`-t [`*n*`]`	Заданный дескриптор *n* открытого файла связан с терминальным устройством; по умолчанию *n* равно `1`
`-u` ***файл***	Заданный ***файл*** существует и установлен его бит смены идентификатора пользователя
`-w` ***файл***	Заданный ***файл*** существует и доступен для записи
`-x` ***файл***	Заданный ***файл*** существует и является исполняемым
f1 `-ef` *f2*	Заданные файлы *f1* и *f2* связаны вместе, т.е. ссылаются на один и тот же файл
f1 `-nt` *f2*	Заданный файл *f1* оказывается более новым, чем файл *f2*
f1 `-ot` *f2*	Заданный файл *f1* оказывается более старым, чем файл *f2*

Условия проверки символьных строк

Ниже перечислены допустимые условия проверки символьных строк.

строка	Указанная ***строка*** не является пустой
`-n` *s1*	Указанная строка *s1* имеет ненулевую длину
`-z` *s1*	Указанная строка *s1* имеет нулевую длину
s1`==`*s2*	Указанные строки *s1* и *s2* одинаковы. В двойных квадратных скобках (`[[ ]]`) строка *s2* может быть подстановочным шаблоном. Чтобы интерпретировать строку *s2* буквально, ее следует заключить в кавычки. Подробнее об этом см. в разделе "Метасимволы подстановки имен файлов", а также описание параметра `nocasematch` в разделе "Параметры оболочки"
s1`=`*s2*	То же, что и операция сравнения `==`. Этим условием следует пользоваться в формах `test` и `[ ]` данной команды ради совместимости со стандартом POSIX и другими оболочками
s1`!=`*s2*	Указанные строки *s1* и *s2* неодинаковы. В двойных квадратных скобках (`[[ ]]`) строка *s2* может быть подстановочным шаблоном. Чтобы интерпретировать строку *s2* буквально, ее следует заключить в кавычки

s1=~s2	Указанная строка ***s1*** совпадает с заданным регулярным выражением ***s2***. Доступно только в двойных квадратных скобках (`[[ ]]`). Для принудительного сопоставления с символьной строкой, а не с регулярным выражением операнд ***s2*** следует заключить в кавычки. Символьные строки, сопоставляемые с подвыражениями в круглых скобках, размещаются в элементах массива **BASH_REMATCH**. Описание переменной массива **BASH_REMATCH** см. в разделе "Встроенные в оболочку переменные", а описание параметров `compat31`, `compat32` и `compat40` — в разделе " Параметры оболочки"
s1<s2	Указанное строковое значение ***s1*** предшествует строковому значению ***s2***. В формах `test` и `[ ]` знак `<` данной команды следует заключить в кавычки, чтобы в оболочке Bash использовался порядок сортировки, принятый на данной машине (обычно в коде ASCII). А в форме `[[ ]]` знак `<` не нужно заключать в кавычки, чтобы в оболочке Bash использовался порядок сортировки с учетом текущих региональных настроек
s1>s2	Указанное строковое значение ***s1*** следует после строкового значения ***s2***. В формах `test` и `[ ]` данной команды знак `>` следует заключить в кавычки, чтобы в оболочке Bash использовался порядок сортировки, принятый на данной машине (обычно в коде ASCII). А в форме `[[ ]]` не нужно заключать знак `>` в кавычки, чтобы в оболочке Bash использовался порядок сортировки с учетом текущих региональных настроек

Внутренние условия

Ниже перечислены условия, которые являются внутренними для оболочки.

`-o` ***параметр***	Заданный ***параметр*** для команды `set -o` установлен
`-R` ***переменная***	Указанной ***переменной*** присвоено значение, являющееся ссылкой на косвенную переменную
`-v` ***переменная***	Указанной ***переменной*** присвоено значение. Она может быть элементом массива

Условия целочисленного сравнения

Ниже перечислены доступные условия целочисленного сравнения.

n1 `-eq` ***n2***	Указанные целые числа ***n1*** и ***n2*** равны
n1 `-ge` ***n2***	Указанное целое число ***n1*** больше или равно целому числу ***n2***
n1 `-gt` ***n2***	Указанное целое число ***n1*** больше целого числа ***n2***
n1 `-le` ***n2***	Указанное целое число ***n1*** меньше целого числа ***n2***
n1 `-lt` ***n2***	Указанное целое число ***n1*** меньше или равно целому числу ***n2***
n1 `-ne` ***n2***	Указанные целые числа ***n1*** и ***n2*** не равны

Комбинированные формы

Ниже перечислены доступные комбинированные формы условий.

(*условие*)

Истинно, если истинно заданное ***условие*** (применяется для группирования). В формах **test** и **[ ]** данной команды следует экранировать круглые скобки знаком \, а в форме **[[ ]]** этого не требуется

! *условие*

Истинно, если ложно заданное ***условие***

***условие*$_1$ -a *условие*$_2$**

Истинно, если истинны оба заданных условия

***условие*$_1$ && *условие*$_2$**

Истинно, если истинны оба заданных условия. Это укороченный вариант предыдущей формы, применяемый только в форме **[[ ]]** данной команды

***условие*$_1$ -o *условие*$_2$**

Истинно, если истинно любое из заданных условий

***условие*$_1$ || *условие*$_2$**

Истинно, если истинно любое из заданных условий. Это укороченный вариант предыдущей формы, применяемый только в форме **[[ ]]** данной команды

Примеры

В приведенных ниже примерах демонстрируется первая строка различных операторов, где может употребляться проверочное условие.

```
while test $# -gt 0                   # Выполнять цикл
                                      # до тех пор,
                                      # пока имеются аргументы
if [ $count -lt 10 ]                  # Если значение переменной
                                      # $count меньше 10
if [ -d .git ]                        # Если существует
                                      # каталог .git
if [ "$answer" != "y" ]               # Если ответ
                                      # не положительный
if [ ! -r "$1" -o ! -f "$1" ]         # Если первый аргумент не
                                      # читается и не является
                                      # обычным файлом
if ! [ -r "$1" ] || ! [ -f "$1" ]     # То же, что и выше
```

time **Определить время выполнения команды**

```
time [-p] [команда]
```

Выполнить заданную *команду* и вывести общее истекшее, пользовательское и системное время (в секундах). Данная команда действует подобно внешней команде `time`, за исключением того, что она позволяет дополнительно определить время выполнения других встроенных команд, а также всех команд, выполняющихся в конвейере. При выводе полученных результатов используется обозначение десятичной точки, принятое в текущих региональных настройках.

В отсутствие заданной *команды* выводится истекшее пользовательское, системное и реальное время для текущей оболочки и ее потомков. Значение переменной `TIMEFORMAT` определяет формат вывода. Подробнее об этом см. оперативную страницу руководства *bash*(1). Если в режиме работы по стандарту POSIX первый аргумент предваряется знаком "минус", оболочка Bash интерпретирует `time` как команду, а не ключевое слово.

Параметр

-p Вывести сводку временных характеристик в формате, определяемом по стандарту POSIX

times **Вывести накопленные величины процессорного времени**

```
times
```

Вывести накопленные величины пользовательского и системного времени для текущей оболочки и выполнявшихся в ней процессов.

trap **Манипулировать сигналами в сценарии оболочки**

```
trap [[команды] сигналы]
trap -l
trap -p
```

Выполнить указанные *команды*, если получен любой из заданных *сигналов*. Вторая форма данной команды служит

для перечисления всех сигналов и их номеров подобно команде `kill -l`, а третья форма — для вывода текущих настроек прерываний в виде, удобном для повторного чтения в дальнейшем. К числу сигналов, которыми можно манипулировать по данной команде, относятся также сигналы, игнорируемые при запуске оболочки, хотя они не могут быть изменены.

К категории общих относятся сигналы `EXIT` (**0**), `HUP` (**1**), `INT` (**2**) и `TERM` (**15**). Если указывается несколько команд, их следует заключить как отдельную группу в кавычки, разделив их в самой группе точками с запятой. Если же *команды* указаны как пустая строка (например, в команде `trap "" сигналы`), то оболочка проигнорирует заданные *сигналы*. А если *команды* вообще опущены, то обработка заданных *сигналов* сводится к действию, выполняемому по умолчанию. И, наконец, если *команды* указаны в виде знака "минус" (**-**), то заданные *сигналы* устанавливаются в свое исходное состояние по умолчанию.

Если же опущены как *команды*, так и *сигналы*, то перечисляются текущие назначения прерываний. См. примеры, приведенные ниже, а также ранее в описании команды `exec`.

Как правило, команда `trap` выводит наименования сигналов, предваряя их префиксом **`SIG`**. Но в режиме работы по стандарту POSIX этот префикс опускается.

ПРИМЕЧАНИЕ

Оболочка не блокирует дополнительно появляющиеся сигналы для выполняющихся прерываний, допуская рекурсивные вызовы прерываний. Поэтому пользуйтесь командой **`trap`** аккуратно!

Совет

Как правило, заданные *сигналы* следует заключать в одиночные кавычки, чтобы отложить подстановки переменных и любых других элементов до тех пор, пока сигнал не будет

обработан. В противном случае подстановки, заключенные в двойные кавычки, будут выполняться раньше, когда выполняется сама команда `trap`.

Сигналы

Для обозначения стандартных сигналов в оболочке Bash допускается указывать номер сигнала или его наименование (как с префиксом **SIG**, так и без него). Кроме того, в оболочке Bash поддерживаются "псевдосигналы", не имеющие настоящие наименования и номера в операционной системе, но предписывающие оболочке выполнить конкретное действие. Такие сигналы перечислены ниже с указанием моментов, когда они появляются.

DEBUG	Выполнение любой команды
ERR	Выдача ненулевого кода завершения
EXIT	Выход из оболочки (обычно по завершении сценария оболочки). Служит также для обозначения выхода из оболочек, запущенных на обработку подстановок
RETURN	Выполнение команды **return** или завершение сценария, запущенного на выполнение по команде **.** (точка) или **source**
0	То же, что и сигнал **EXIT**, но служит для исторической совместимости с оболочкой Bourne shell

Примеры

```
trap "" INT       # Проигнорировать прерывания (по сигналу 2)
trap INT          # Снова перехватывать прерывания
                  # по данному сигналу
```

Удалить временный файл, заданный в переменной `$tmp`, когда завершается программа оболочки, пользователь выходит из системы, нажимает комбинацию клавиш <Ctrl+C> или выполняет команду `kill`:

```
trap "rm -f $tmp; exit" EXIT HUP INT TERM    # Обозначение
                                             # сигналов
                                             # по стандарту
POSIX
trap "rm -f $tmp; exit" 0 1 2 15             # Первоначальное
                                             # обозначение
                                             # сигналов
                                             # в оболочке
```

Вывести сообщение об очистке, когда программа оболочки получит сигнал SIGHUP, SIGINT или SIGTERM:

```
trap 'echo Interrupt! Cleaning up...' HUP INT TERM
```

---

true — **Выйти с возвратом истинного значения, обозначающего удачное завершение**

```
true
```

Встроенная команда, завершающаяся возвратом истинного значения, обозначающего удачный исход выполнения.

---

type — **Вывести тип команды**

type [-afpPt] *команды*

Показать, является ли указанная *команда* внешней, встроенной, псевдонимом, ключевым словом оболочки или определенной в ней функцией.

Параметры

- **-a** Вывести все местоположения, заданные в переменной окружения **$PATH**, включая указанные ***команды***, псевдонимы и функции. Чтобы исключить из вывода псевдонимы и функции, следует указать оба параметра: **-a** и **-p**
- **-f** Исключить из вывода поиск функций, как по команде **command**
- **-p** Если по команде **type -t** указанная ***команда*** выводится вместе со словом **file**, вывести полный путь к соответствующему исполняемому файлу. В противном случае ничего не выводить
- **-P** То же, что и параметр **-p**, но в данном случае произвести принудительный поиск по содержимому переменной окружения **$PATH**, даже если бы по команде **type -t** не было выведено слово **file**
- **-t** Вывести слово, описывающее каждую указанную ***команду***. Это может быть слово **alias**, **builtin**, **file**, **function** или **keyword** в зависимости от типа каждой указанной ***команды***

Пример

```
$ type mv read if
mv is /bin/mv                  # mv — это команда, доступная
                               # по пути /bin/mv
read is a shell builtin        # read — это встроенная команда
if is a shell keyword          # if — это ключевое слово оболочки
```

`typeset` Объявить переменные оболочки и манипулировать их атрибутами

typeset [*параметры*] [*переменная*[=*значение* ...]]

Данная команда действует подобно команде declare. См. приведенное ранее описание команды declare.

`ulimit` Накладывать различные ограничения на процессы

ulimit [*параметры*] [*n*]

Вывести значение одного или нескольких ограничений, накладываемых на ресурсы, или наложить на ресурс ограничение *n*, если оно задано. На ресурсы могут быть наложены жесткие (параметр **-H**) или мягкие (параметр **-S**) ограничения. По умолчанию командой ulimit накладываются оба вида ограничений или выводятся мягкие ограничения. Отдельные параметры определяют, на какие именно ресурсы накладываются ограничения.

Параметры

-H Жесткое ограничение. Любой пользователь может ослабить жесткое ограничение, тогда как усилить его могут только привилегированные пользователи

-S Мягкое ограничение. Оно должно быть меньше или равно жесткому ограничению

-a Вывести все ограничения

-b Максимальный размер буфера сокета

-c Максимальный размер файлов ядра. По умолчанию обозначается блоками величиной 1 Кбайт в качестве единиц измерения. А в режиме работы по стандарту POSIX обозначается блоками величиной 512 байт

-d Максимальный размер сегмента данных или динамической памяти (так называемой "кучи") в килобайтах

-e Максимальный приоритет планирования (приоритетное значение)

-f Максимальный размер файлов (этот параметр устанавливается по умолчанию). В качестве единиц измерения по умолчанию установлены блоки 1 Кбайт. А в режиме работы по стандарту POSIX обозначается блоками величиной 512 байт

-i Максимальный номер ждущего сигнала

-k Максимальное количество очередей типа kqueue. (Действует не во всех системах)

`-l` Максимальный размер адресного пространства, которое может быть заблокировано в оперативной памяти

`-m` Максимальный объем физической памяти в килобайтах. (Имеется не во всех системах.)

`-n` Максимальное количество дескрипторов файлов

`-p` Размер буферов каналов. (Имеется не во всех системах.)

`-q` Максимальное количество байтов в очередях сообщений по стандарту POSIX

`-r` Максимальный приоритет планирования в реальном времени

`-s` Максимальный размер сегмента стека в килобайтах

`-t` Максимальная величина процессорного времени в секундах

`-T` Максимальное количество потоков исполнения

`-u` Максимальное количество процессов, которые могут принадлежать одному пользователю

`-v` Максимальный объем виртуальной памяти в килобайтах. (Имеется не во всех системах.)

`-x` Максимальное количество блокировок файлов

---

umask **Отобразить или установить маску создания файла в процессе**

```
umask [nnn]
umask [-pS] [маска]
```

Отобразить или установить маску создания файла по заданному восьмеричному значению *nnn*. Маска создания файла определяет, какие именно биты разрешений сбрасываются (например, по команде `umask 002` получаются разрешения `rw-rwr--` на доступ к файлу). Во второй форме данной команды символическая *маска* обозначает сохраняемые разрешения на доступ к файлу.

Параметры

`-p` Вывести полученный результат в таком виде, который позволяет в дальнейшем прочитать его повторно средствами оболочки

`-S` Вывести текущую маску, используя символическое обозначение

---

unalias **Удалить определенные ранее псевдонимы**

```
unalias имена
unalias -a
```

Удалить указанные *имена* из списка псевдонимов. См. также приведенное ранее описание команды `alias`.

Параметр

-a Удалить все псевдонимы

---

unset **Удалить переменные или функции**

`unset [`*параметры*`]` *имена*

Удалить определения функций или переменных, *имена* которых перечислены в командной строке. Если *имя* указано с индексом (например, в команде `unset foo[2]`), то в исходное состояние устанавливается соответствующий элемент массива. По нулевому индексу в исходное состояние устанавливается соответствующая скалярная переменная.

Параметры

-f Удалить функции по указанным ***именам***

-n Удалить косвенные переменные по указанным ***именам***. См. также раздел "Косвенные переменные"

-v Удалить переменные по указанным ***именам*** (устанавливается по умолчанию)

---

until **Синтаксис для организации цикла, выполняющегося до тех пор, пока заданное условие не станет истинным**

```
until условие
do
    команды
done
```

Выполнять указанные *команды* до тех пор, пока удовлетворяется *условие*, которое обычно задается с помощью команды `test`. Соответствующие примеры см. в приведенных выше описаниях команд `case` и `test`.

---

wait **Ожидать завершения процесса или задания**

`wait [-n] [`*идентификатор*`]`

В отсутствие параметра или аргументов приостановить выполнение до тех пор, пока не завершатся все фоновые задания,

а затем возвратить нулевой код завершения. Если же указан *идентификатор*, то приостановить выполнение до тех пор, пока не завершится процесс или задание с данным *идентификатором*, а затем возвратить код его завершения. Следует, однако, иметь в виду, что переменная оболочки `$!` содержит идентификатор самого последнего фонового процесса.

В отсутствие аргументов поведение команды `wait` зависит от установки режима работы по стандарту POSIX. Как правило, по команде `wait` ожидается завершение всех фоновых процессов, а затем выполняется прерывание `SIGCHLD` столько раз, сколько завершилось процессов. А в режиме работы по стандарту POSIX завершающийся порожденный процесс прерывает выполнение команды `wait`, принуждая ее завершиться с кодом **128 + SIGCHLD**. Оболочка Bash попытается выполнить обработчик прерываний **SIGCHLD** для каждого порожденного процесса, но при этом не гарантируется, что она это сделает.

Параметр

-n Ожидать завершения любого задания и возвратить код его завершения

Пример

```
wait $!             # Ожидать завершения самого
                    # последнего фонового процесса
```

---

while **Синтаксис для организации цикла, выполняющегося до тех пор, пока заданное условие остается истинным**

```
while условие
do
    команды
done
```

Выполнять указанные *команды* до тех пор, пока удовлетворяется *условие*, которое обычно задается с помощью команды `test`. Соответствующие примеры см. в приведенных выше описаниях команд `case` и `test`.

Дополнительные источники информации

В этом разделе вкратце описываются другие источники информации по оболочке Bash и на связанные с ней темы.

Оперативно доступные ресурсы

Ниже перечислены оперативно доступные ресурсы по оболочке Bash.

- **http://ftp.gnu.org/gnu/bash**. Каталог верхнего уровня для выпусков исходного кода оболочки Bash. Как правило, исходный код доступен в виде архивных файлов с расширением `.tar.gz`, например `bash-4.4.tar.gz`.
- **ftp://ftp.gnu.org/pub/gnu/bash/bash-4.4-patches**. В этом каталоге находятся "заплаты" с исправлениями для версии Bash 4.4.
- **http://www.gnu.org/software/bash/bash.html** и
- **https://tiswww.case.edu/php/chet/bash/bashtop.html**.По этим адресам указаны две начальные страницы, посвященные оболочке Bash.
- **http://bashdb.sourceforge.net**. Эта страница посвящена отладчику оболочки Bash.
- **http://bash-completion.alioth.debian.org/**. На этой странице можно найти накопленные Ианом Макдональдом спецификации автозавершения команд, вводимых в оболочке Bash.
- **http://www.gnu.org/software/bash/manual/html_node/Bash-POSIX-Mode.html**. На этой странице доступна полная документация, описывающая все последствия работы в режиме по стандарту POSIX. Многие отличия едва заметны и не оказывают особого влияния на применение оболочки Bash в настоящее время.

- **`http://www.opengroup.org/onlinepubs/9699919799`**. На этой странице находится оперативно доступная версия стандарта POSIX.
- **`http://tobold.org/article/rc`**. На этой странице находятся загружаемые ресурсы оболочки `rc` для систем Unix.

Литература

Ниже перечислена литература, рекомендованная для дальнейшего изучения оболочки Bash.

- Newham, Cameron. *Learning the bash Shell*, Third Edition. Sebastopol: O'Reilly Media, 2005.
- Robbins, Arnold, and Nelson H.F. Beebe. *Classic Shell Scripting*. Sebastopol: O'Reilly Media, 2005.

Предметный указатель

М

О

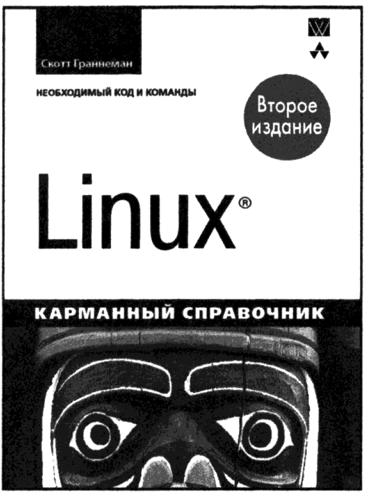
П

LINUX

Карманный справочник

Второе издание

Скотт Граннеман



Книга представляет собой краткий справочник по основным командам операционной системы Linux. В книге содержится множество готовых к использованию фрагментов программ и команд для выполнения типичных задач в системе Linux. Книга будет полезна как для новичков, только приступающих к изучению Linux, так и для опытных пользователей, применяющих оболочку для решения разных задач: от администрирования до программирования.

www.williamspublishing.com

ISBN 978-5-8459-2101-7 **в продаже**

O'REILLY

Bash. Карманный справочник

Чтобы научиться искусно взаимодействовать с Mac OS X, Linux и прочими Unix-подобными операционными системами, нужно овладеть навыками работы с оболочкой Bash. И этот краткий справочник позволит вам иметь под рукой самые основные сведения о Bash.

В нем вы сможете быстро найти ответы на насущные вопросы, которые зачастую возникают при написании сценариев оболочки, включая следующие: какие символы следует заключать в кавычки, как выполнять подстановку переменных и правильно пользоваться массивами? Настоящее издание, обновленное по версии Bash 4.4, в удобной и краткой форме дает ответы на эти и многие другие вопросы.

Основные темы книги

- Вызов командной оболочки
- Синтаксис языка оболочки
- Функции и переменные
- Предыстория команд
- Автозавершение вводимых команд
- Управление заданиями
- Выполнение команд
- Сопроцессы
- Ограниченные оболочки
- Встроенные команды

Арнольд Роббинс — профессиональный программист и автор технической литературы, работающий с системами Unix с 1980 года. Кроме того, он является одним из авторов второго издания книги *Classic Shell Scripting*, а также автором четвертого издания книги *Effective awk Programming* и ряда других книг, вышедших в издательстве O'Reilly. В настоящее время Арнольд занимается сопровождением версии GNU языка Awk (`gawk`) и документации на него.

Диалектика

http://www.williamspublishing.com

www.oreilly.com

Категория: операционные системы Unix, Linux
Предмет рассмотрения: оболочка Bash
Уровень: промежуточный — опытный

ISBN 978-5-9909445-4-1

17027

9 785990 944541